1 2-Sat

```
int n;
vector<vector<int>> adj, adj_t;
vector<bool> used,assignment;
vector<int> order, comp;
void dfs1(int v) {
 used[v] = true;
 for (int u : adj[v]) { if (!used[u])
   dfs1(u);
 }
 order.push_back(v);
}
void dfs2(int v, int cl) {
 comp[v] = cl;
 for (int u : adj_t[v]) { if (comp[u] == -1)
   dfs2(u, cl);
 }
}
bool solve_2SAT() {
 order.clear(); used.assign(n, false);
 for (int i = 0; i < n; ++i) { if (!used[i])
   dfs1(i);
 }
 comp.assign(n, -1);
 for (int i = 0, j = 0; i < n; ++i) {
  int v = order[n - i - 1];
  if (comp[v] == -1) dfs2(v, j++);
 }
 assignment.assign(n / 2, false);
 for (int i = 0; i < n; i += 2) {
  if (comp[i] == comp[i + 1]) return false;
  assignment[i / 2] = comp[i] > comp[i + 1];
 }
 return true;
}
void add_disjunction(int a, bool na, int b, bool nb) {
 // na and nb signify whether a and b are to be negated
 a = 2*a ^ na; b = 2*b ^ nb;
 int neg_a = a ^ 1; int neg_b = b ^ 1;
 adj[neg_a].push_back(b); adj[neg_b].push_back(a);
 adj_t[b].push_back(neg_a); adj_t[a].push_back(neg_b);
}
```

2 Aho

```
const int K = 26;

class Node{
public:
 vector<int>nxt;
 vector<int>go;
 int suf ;
 int ssuf;
 bool leaf;
 int parent;
 int ch;
 Node(int p=-1,int ch = -1) {
  this->parent = p;
  this->ch = ch;
  nxt.resize(K,-1);
  go.resize(K,-1);
  leaf = false;
  suf = -1;
  ssuf = -1;
 }
};

class AhoCorasick{
public:
 vector<Node>nodes;

 AhoCorasick() {
  nodes.emplace_back(Node());
 }

 int get_num(char ch) {
  int ret = ch-'a';
  return ret;
 }

 void add_string(string &s) {
  int v =0;
  for(int i=0; i<s.size(); i++) {
   int now = get_num(s[i]);
   if(nodes[v].nxt[now]==-1) {
    nodes[v].nxt[now] = nodes.size();
    nodes.emplace_back(v,now);
   }
   v = nodes[v].nxt[now];
  }
  nodes[v].leaf = true;
 }
 int go(int v, int ch) {
  int &ret = nodes[v].go[ch];
  if(ret!=-1)
   return ret;
  if(nodes[v].nxt[ch]!=-1) {
   return ret = nodes[v].nxt[ch];
  }
  if(v==0) {
   return ret = 0;
  }
  return ret = go(get_link(v), ch);
 }
 int get_link(int v) {
  int &ret = nodes[v].suf;
  if(ret!=-1)
   return ret;
  if(v==0 || nodes[v].parent==0) {
   return ret =0;
  }
  return ret = go(get_link(nodes[v].parent) , nodes[v].ch);
 }
 int exit_link(int v) {
  int &ret = nodes[v].ssuf;
  if(ret!=-1)
   return ret;
  if(v==0 || nodes[v].parent==0) {
   return ret = 0;
  }
  int s = get_link(v);
  if(nodes[s].leaf) {
   return ret = s;
  }
  return ret = exit_link(s);
 }

 void feed(string &s) {
  int v =0;
  for(int i=0;i<s.size();i++) {
   int ch = get_num(s[i]);
   v = go(v,ch);
   int u =v ;
   while(u!=0) {
    u = exit_link(u);
   }
  }
 }

};
```

3 ArticulationBridge

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

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

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

4 ArticulationPoint

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

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

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

5 BIT2D

```
struct FenwickTree2D {
 vector<vector<int>> bit;
 int n, m;

 // init(...) { ... }

 int sum(int x, int y) {
  int ret = 0;
  for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
   for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
    ret += bit[i][j];
  return ret;
 }

 void add(int x, int y, int delta) {
  for (int i = x; i < n; i = i | (i + 1))
   for (int j = y; j < m; j = j | (j + 1))
    bit[i][j] += delta;
 }
};
```

6 Berlekamp

```
#define SZ 233333
const int MOD=1e9+7;
ll qp(ll a,ll b){
 ll x=1; a%=MOD;
 while(b) {
  if(b&1) x=x*a%MOD;
  a=a*a%MOD; b>>=1;
 }
 return x;
```

```
}
namespace linear_seq{
inline vector<int> BM(vector<int> x){
 vector<int> ls,cur;
 int lf,ld;
 for(int i=0; i<int(x.size()); ++i) {
  ll t=0;
  for(int j=0; j<int(cur.size()); ++j)
   t=(t+x[i-j-1]*(ll)cur[j])%MOD;
  if((t-x[i])%MOD==0) continue;
  if(!cur.size()) {
   cur.resize(i+1); lf=i; ld=(t-x[i])%MOD;
   continue;
  }
  ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD;
  vector<int> c(i-lf-1);
  c.push_back(k);
  for(int j=0; j<int(ls.size()); ++j)
       c.push_back(-ls[j]*k%MOD);
  if(c.size()<cur.size()) c.resize(cur.size());
  for(int j=0; j<int(cur.size()); ++j) c[j]=(c[j]+cur[j])%MOD;
  if(i-lf+(int)ls.size()>=(int)cur.size())
   ls=cur,lf=i,ld=(t-x[i])%MOD;
  cur=c;
 }
 for(int i=0; i<int(cur.size()); ++i)
      cur[i]=(cur[i]%MOD+MOD)%MOD;
 return cur;
}
int m;
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
inline void mull(ll*p,ll*q){
 for(int i=0; i<m+m; ++i) t_[i]=0;
 for(int i=0; i<m; ++i) if(p[i])
   for(int j=0; j<m; ++j)
    t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
 for(int i=m+m-1; i>=m; --i) if(t_[i])
   for(int j=m-1; ~j; --j)
    t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
 for(int i=0; i<m; ++i) p[i]=t_[i];
}
inline ll calc(ll K){
 for(int i=m; ~i; --i) s[i]=t[i]=0;
 s[0]=1;
 if(m!=1) t[1]=1; else t[0]=h[0];
 while(K){if(K&1)mull(s,t);mull(t,t);K>>=1;}
 ll su=0;
 for(int i=0; i<m; ++i) su=(su+s[i]*a[i])%MOD;
 return (su%MOD+MOD)%MOD;
}
inline int work(vector<int> x,ll n){
 if(n<int(x.size())) return x[n];
 vector<int> v=BM(x); m=v.size();
 if(!m) return 0;
 for(int i=0; i<m; ++i) h[i]=v[i],a[i]=x[i];
 return calc(n);
}
}
using linear_seq::work;
```

7 CHT

```
bool Q;

struct Line{
 mutable ll k, m, p; // slope, y-intercept, last optimal x
 bool operator<(const Line& o) const {
  return Q ? p < o.p : k < o.k;
 }
};

struct LineContainer : multiset<Line>{
 const ll inf = LLONG_MAX;
 ll div(ll a, ll b) // floored division {
  if (b < 0)
   a *= -1, b *= -1;
  if (a >= 0)
   return a / b;
  return -((-a + b - 1) / b);
 }
 // updates x->p, determines if y is unneeded
 bool isect(iterator x, iterator y) {
  if (y == end()) {
   x->p = inf;
   return 0;
  }
  if (x->k == y->k)
   x->p = x->m > y->m ? inf : -inf;
  else
   x->p = div(y->m - x->m, x->k - y->k);
  return x->p >= y->p;
 }

 void add(ll k, ll m) {
  auto z = insert({k, m, 0}), y = z++, x = y;
  while (isect(y, z))
   z = erase(z);
  if (x != begin() && isect(--x, y))
   isect(x, y = erase(y));
  while ((y = x) != begin() && (--x)->p >= y->p)
   isect(x, erase(y));
 }

 ll query(ll x) // gives max value {
  assert(!empty());
  Q = 1;
  auto l = *lower_bound({0, 0, x});
  Q = 0;
  return l.k * x + l.m;
 }
};
```

8 CRT

```
/** Works for non-coprime moduli.
 Returns {0,0} if solution does not exist or input is invalid.
 Otherwise, returns {x,L}, where x is the solution unique to
      mod L
*/
constexpr long long safe_mod(long long x, long long m) {
 x %= m;
 if (x < 0) x += m;
 return x;
}
constexpr std::pair<long long, long long> inv_gcd(long long a,
     long long b) {
 a = safe_mod(a, b);
 if (a == 0) return {b, 0};
 long long s = b, t = a;
 long long m0 = 0, m1 = 1;
 while (t) {
  long long u = s / t;
  s -= t * u;
  m0 -= m1 * u;
  auto tmp = s;
  s = t;
  t = tmp;
  tmp = m0;
  m0 = m1;
  m1 = tmp;
 }
 if (m0 < 0) m0 += b / s;
 return {s, m0};
}

std::pair<long long, long long> crt(const std::vector<long
     long>& r,
         const std::vector<long long>& m) {
 assert(r.size() == m.size());
 int n = int(r.size());
 long long r0 = 0, m0 = 1;
 for (int i = 0; i < n; i++) {
  assert(1 <= m[i]);
  long long r1 = safe_mod(r[i], m[i]), m1 = m[i];
  if (m0 < m1) {
   std::swap(r0, r1);
   std::swap(m0, m1);
  }
  if (m0 % m1 == 0) {
   if (r0 % m1 != r1) return {0, 0};
   continue;
  }
  long long g, im;
  std::tie(g, im) = inv_gcd(m0, m1);
  long long u1 = (m1 / g);
  if ((r1 - r0) % g) return {0, 0};
  long long x = (r1 - r0) / g % u1 * im % u1;
  r0 += x * m0;
  m0 *= u1;
  if (r0 < 0) r0 += m0;
 }
 return {r0, m0};
}
```

9 Centroid

```
class CentroidDecomposition{
public:
 vector<vector<int> >adj;
 vector<int>sz,parent;
 vector<bool>vis;
 int n,root;
 CentroidDecomposition(vector<vector<int> >&a) {
  adj = a; n = adj.size(); sz.resize(n); parent.resize(n,-1);
  vis.resize(n,false);
  build(0,-1);
 }
 void build(int s, int p) {
  dfs(s,p);
  int c= centroid(s,p,sz[s]); vis[c]= true;
  if(p!=-1) { parent[c] = p; }
  else { root = c; }
  for(auto it:adj[c]) { if(vis[it]) continue;
   build(it,c);
  }
 }
 void dfs(int s, int p) {
  sz[s] = 1;
  for(auto i:adj[s]) { if(i==p || vis[i]) continue;
   dfs(i,s); sz[s]+= sz[i];
  }
 }
 int centroid(int s,int p, int total) {
  for(auto i:adj[s]) { if(i==p || vis[i]) continue;
   if(sz[i]*2>total) return centroid(i,s,total);
  }
  return s;
 }
};
```

10 ContinuedFraction

```
/**
 * Description: Given $f$ and $N$, finds the smallest fraction
      $p/q \in [0, 1]$
 * such that $f(p/q)$ is true, and $p, q less than N$.
 * You may want to throw an exception from $f$ if it finds an
      exact solution,
 * in which case $N$ can be removed.
 * Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); //
      {1,3}
 * Time: O(\log(N))
 * Status: fuzz-tested for n <= 300
 */

typedef __int128_t lll;

struct Frac{
 lll p, q;
};

template<class F>
Frac fracBS(F f, lll N){
 bool dir = 1, A = 1, B = 1;
 Frac lo{0, 1}, hi{1, 0};
 if (f(lo))
  return lo;
 assert(f(hi));
 while (A || B) {
  lll adv = 0, step = 1;
  for (int si = 0; step; (step *= 2) >>= si) {
   adv += step;
```

```
   Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
   if ((mid.p < 0 ? -mid.p : mid.p) > N || mid.q > N || dir ==
       !f(mid)) {
    adv -= step;
    si = 2;
   }
  }
  hi.p += lo.p * adv;
  hi.q += lo.q * adv;
  dir = !dir;
  swap(lo, hi);
  A = B;
  B = !!adv;
 }
 return dir ? hi : lo;
}

int main(){
 Frac target = fracBS([&](Frac fr) {
  if (fr.q == 0)
   return true;
  return minfr < fr;
 }, mid);
}
```

11 DC DP

```
int dp_before[N],dp_cur[N],cnt[N],l=0,r=-1,cur;
void add(int idx){
 cnt[a[idx]]++; if(cnt[a[idx]]==1)cur++;
}
void remove(int idx){
 cnt[a[idx]]--; if(cnt[a[idx]]==0) cur--;
}
int cost(int L,int R){
 while (l > L) { l--; add(l); }
 while (r < R) { r++; add(r); }
 while (l < L) { remove(l); l++; }
 while (r > R) { remove(r);r--;}
 return cur;
}
void compute(int l, int r, int optl, int optr){
 if (l > r) return;
 int mid = (l + r) >> 1;
 pair<int, int> best = {-1e9, -1};
 for (int k = optl; k <= min(mid, optr); k++) {
  if(best.second==-1) { best = {dp_before[k] +cost(k+1, mid),
      k};}
  else { best = max(best, {dp_before[k] +cost(k+1, mid), k}); }
 }
 dp_cur[mid] = best.first; int opt = best.second;
 compute(l, mid - 1, optl, opt);compute(mid + 1, r, opt, optr);
}
```

12 DEBUG TEMPLATE

```
void err(istream_iterator<string> it) {cout<<endl;}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args){
 cout << *it << " = " << a << " ";err(++it, args...);
}
template<class T1, class T2>
ostream &operator <<(ostream &os, pair<T1,T2>&p) {
 os<<"{"<<p.first<<", "<<p.second<<"} ";
 return os;
}
#define debug(args...) { string _s = #args;
     replace(_s.begin(), _s.end(), ',', ' '); stringstream
     _ss(_s); istream_iterator<string> _it(_ss); err(_it,
     args); }
```

13 DSU Rollback

```
struct dsu_save {
 int v, rnkv, u, rnku;
 dsu_save() {}
 dsu_save(int _v, int _rnkv, int _u, int _rnku)
  : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};
struct dsu_with_rollbacks {
 vector<int> p, rnk;
 int comps;
 stack<dsu_save> op;
 dsu_with_rollbacks() {}
 dsu_with_rollbacks(int n) {
  p.resize(n);
  rnk.resize(n);
  for (int i = 0; i < n; i++) {
   p[i] = i;
   rnk[i] = 0;
  }
  comps = n;
 }
 int find_set(int v) {
  return (v == p[v]) ? v : find_set(p[v]);
 }
 bool unite(int v, int u) {
  v = find_set(v);
  u = find_set(u);
  if (v == u)
   return false;
  comps--;
  if (rnk[v] > rnk[u])
   swap(v, u);
  op.push(dsu_save(v, rnk[v], u, rnk[u]));
  p[v] = u;
  if (rnk[u] == rnk[v])
   rnk[u]++;
  return true;
 }
 void rollback() {
  if (op.empty())
   return;
  dsu_save x = op.top();
  op.pop();
  comps++;
  p[x.v] = x.v;
  rnk[x.v] = x.rnkv;
  p[x.u] = x.u;
  rnk[x.u] = x.rnku;
 }
};
struct query {
 int v, u;
 bool united;
 query(int _v, int _u) : v(_v), u(_u) {
 }
};
struct QueryTree {
 vector<vector<query>> t;
 dsu_with_rollbacks dsu;
 int T;
 QueryTree() {}
 QueryTree(int _T, int n) : T(_T) {
  dsu = dsu_with_rollbacks(n);
  t.resize(4 * T + 4);
 }
 void add_to_tree(int v, int l, int r, int ul, int ur, query&
      q) {
  if (ul > ur)
   return;
  if (l == ul && r == ur) {
   t[v].push_back(q);
   return;
  }
  int mid = (l + r) / 2;
  add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
  add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q);
 }
 void add_query(query q, int l, int r) {
  add_to_tree(1, 0, T - 1, l, r, q);
 }
 void dfs(int v, int l, int r, vector<int>& ans) {
  for (query& q : t[v]) {
   q.united = dsu.unite(q.v, q.u);
  }
  if (l == r)
   ans[l] = dsu.comps;
  else {
   int mid = (l + r) / 2;
   dfs(2 * v, l, mid, ans);
   dfs(2 * v + 1, mid + 1, r, ans);
  }
  for (query q : t[v]) {
   if (q.united)
    dsu.rollback();
  }
 }
 vector<int> solve() {
  vector<int> ans(T);
  dfs(1, 0, T - 1, ans);
  return ans;
 }
};
```

14 Dinic

```
const long long flow_inf = 1e18;
struct FlowEdge {
  int v,u,id; long long cap, flow = 0;
  FlowEdge(int v, int u, long long cap,int id=-1) : v(v),
       u(u), cap(cap),id(id) {}
};
struct Dinic
{
  vector<FlowEdge> edges; vector<vector<int> > adj;
  int n, m = 0; int s, t;
  vector<int> level, ptr,flow_through;
  queue<int> q; vector<bool>vis;
  int maxid=0;
  Dinic() {}
  Dinic(int n) : n(n) {
    vis.resize(n); adj.resize(n);
    level.resize(n); ptr.resize(n);
  }
  void add_edge(int v, int u, long long cap,int id=-1) {
    edges.emplace_back(v, u, cap,id);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
    if(id!=-1)maxid++;
  }
  void dfs2(int s) {
    vis[s] = 1;
    for(int i:adj[s]) {
      int id = i; int u = edges[id].v;
      int v = edges[id].u;
      if(edges[id].flow!=edges[id].cap && !vis[v])
      {
        dfs2(v);
      }
    }
  }
  vector<int> getMinCut() {
    dfs2(s); vector<int>ret;
    for(int i=0; i<n; i++) {
      if(vis[i]) ret.push_back(i);
    }
    return ret;
  }
  bool bfs() {
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int id : adj[v])
      {
        if (edges[id].cap - edges[id].flow < 1)
          continue;
        if (level[edges[id].u] != -1)
          continue;
        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);
      }
    }
    return level[t] != -1;
  }
  long long dfs(int v, long long pushed) {
    if (pushed == 0) return 0;
```

```
    if (v == t) return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++){
      int id = adj[v][cid]; int u = edges[id].u;
      if (level[v] + 1 != level[u] || edges[id].cap -
          edges[id].flow < 1)
        continue;
      long long tr = dfs(u, min(pushed, edges[id].cap -
          edges[id].flow));
      if (tr == 0)
        continue;
      edges[id].flow += tr; edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }
  long long flow(int _s,int _t) {
    s=_s; t=_t;
    long long f = 0;
    while (true)
    {
      fill(level.begin(), level.end(), -1);
      level[s] = 0; q.push(s);
      if (!bfs()) break;
      fill(ptr.begin(), ptr.end(), 0);
      while (long long pushed = dfs(s, flow_inf)){
        f += pushed;
      }
    }
    flow_through.assign(maxid+1, 0);
    for(int i = 0; i < n; i++){
      for(auto j : adj[i]) {
        int idx = j;
        FlowEdge e = edges[idx];
        if(e.id >= 0)flow_through[e.id] = e.flow;
      }
    }
    return f;
  }
};

/*for bipartite graph*/
class Minimum_node_cover
{
public:
  map<pair<int,int>,bool>matched;
  vector<vector<int> >adj;
  vector<int>minimum_vertex,maximum_set,l,r;
  vector<bool>vis;
  /*number of nodes in dinic without source and destination src
      = 0 ,dest = sz+1
d.flow() should be called before constructor calling*/
  Minimum_node_cover(int sz, Dinic &d){
    adj.resize(sz+5);vis.resize(sz+5);
    for(auto it:d.edges){
      if(it.u>0 && it.u <=sz && it.v>0 && it.v<=sz &&
          it.cap==1){
        if(it.flow==1){
          adj[it.u].push_back(it.v);
          matched[ {it.u,it.v}]=1;
        }
        else adj[it.v].push_back(it.u);
      }
    }
    for(auto it:d.edges){
      if(it.v==0 && it.cap==1) l.push_back(it.u);
      if(it.u==sz+1 && it.cap==1) r.push_back(it.v);
    }
    sort(l.begin(),l.end());sort(r.begin(),r.end());
    l.resize(distance(l.begin(),unique(l.begin(),l.end())) );
    r.resize(distance(r.begin(),unique(r.begin(),r.end())) );
    for(auto it:d.edges){
      if(it.v==0 && it.cap==1 && it.flow==0){
        if(!vis[it.u]) dfs2(it.u, 1);
      }
    }
    for(int i:l){
      if(!vis[i]) minimum_vertex.push_back(i);
      else maximum_set.push_back(i);
    }
    for(int i:r){
      if(vis[i]) minimum_vertex.push_back(i);
      else maximum_set.push_back(i);
    }
  }
  void dfs2(int s, bool bam){
    vis[s] = 1;
    if(bam){
      for(int i:adj[s]){
        if(vis[i]) continue;
        if(matched[ {s,i}]==0) dfs2(i,0);
      }
    }
    else{
      for(int i:adj[s]){
        if(vis[i]) continue;
        if(matched[{s,i}]==1) dfs2(i,1);
      }
    }
  }
};
//flow_through[i] = extra flow beyond 'low' sent through edge i
struct LR_Flow{
  Dinic F;int n, s, t;
  struct edge{
    int u, v, l, r, id;
  };
  vector<edge> edges;
  LR_Flow() {}
  LR_Flow(int _n){
    n = _n + 2;s = n - 2, t = n - 1;;
    edges.clear();
  }
  void add_edge(int u, int v, int l, int r, int id = -1){
    assert(0 <= l && l <= r);
    edges.push_back({u, v, l, r, id});
  }
  bool feasible(int _s = -1, int _t = -1, int L = -1, int R =
       -1)
  {
    if (L != -1)
      edges.push_back({_t, _s, L, R, -1});
    F = Dinic(n);
    long long target = 0;
    for (auto e : edges){
      int u = e.u, v = e.v, l = e.l, r = e.r, id = e.id;
      if (l != 0){
        F.add_edge(s, v, l);F.add_edge(u, t, l);
        target += l;
      }
      F.add_edge(u, v, r - l, id);
    }
    auto ans = F.flow(s, t);
    if (L != -1)edges.pop_back();
    if (ans < target)return 0; //not feasible
    return 1;
  }
  int max_flow(int _s, int _t){ //-1 means flow is not feasible
    int mx = 1e5 + 9;
    if (!feasible(_s, _t, 0, mx))return -1;
    return F.flow(_s, _t);
  }
  int min_flow(int _s, int _t){ //-1 means flow is not feasible
    int mx = 1e9;int ans = -1, l = 0, r = mx;
    while (l <= r){
      int mid = l + r >> 1;
      if (feasible(_s, _t, 0, mid))
        ans = mid, r = mid - 1;
      else l = mid + 1;
    }
    return ans;
  }
};
```

15 Diophantine

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

bool find_any_solution(int a, int b, int c, int &x0, int &y0,
     int &g) {
 g = gcd(abs(a), abs(b), x0, y0);
 if (c % g) {
  return false;
 }

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

int find_all_solutions(int a, int b, int c, int minx, int
     maxx, int miny, int maxy) {
 int x, y, g;
 if (!find_any_solution(a, b, c, x, y, g))
  return 0;
 a /= g;
 b /= g;

 int sign_a = a > 0 ? +1 : -1;
 int sign_b = b > 0 ? +1 : -1;

 shift_solution(x, y, a, b, (minx - x) / b);
 if (x < minx)
  shift_solution(x, y, a, b, sign_b);
 if (x > maxx)
  return 0;
 int lx1 = x;

 shift_solution(x, y, a, b, (maxx - x) / b);
 if (x > maxx)
  shift_solution(x, y, a, b, -sign_b);
 int rx1 = x;

 shift_solution(x, y, a, b, -(miny - y) / a);
 if (y < miny)
  shift_solution(x, y, a, b, -sign_a);
 if (y > maxy)
  return 0;
 int lx2 = x;

 shift_solution(x, y, a, b, -(maxy - y) / a);
 if (y > maxy)
  shift_solution(x, y, a, b, sign_a);
 int rx2 = x;

 if (lx2 > rx2)
  swap(lx2, rx2);
 int lx = max(lx1, lx2);
 int rx = min(rx1, rx2);

 if (lx > rx)
  return 0;
 return (rx - lx) / abs(b) + 1;
}
```

16 DiscreteLog

```
// Returns minimum x for which a ^ x % m = b % m. O(sqrt(m) )
int solve(int a, int b, int m){
// if (a == 0)
//  return b == 0 ? 1 : -1;
 a %= m, b %= m;int 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, ++add; k = (k * 1ll * a / g) % m;
```

```
 }
 int n = sqrt(m) + 1; int an = 1;
 for (int i = 0; i < n; ++i) an = (an * 1ll * a) % m;
 unordered_map<int, int> vals;
 for (int q = 0, cur = b; q <= n; ++q) {
  vals[cur] = q; cur = (cur * 1ll * a) % m;
 }
 for (int p = 1, cur = k; p <= n; ++p) {
  cur = (cur * 1ll * an) % m;
  if (vals.count(cur)) { int ans = n * p - vals[cur] + add;
   return ans;
  }
 }
 return -1;
}
```

17 EulerTour

```
vector<multiset<int> >adj; vector<int>ans;
void euler_circuit(int src){
 stack<int>st; st.push(src);
 while(!st.empty()){
  int v = st.top();
  if(adj[v].size()==0){
   ans.push_back(v); st.pop();
  }
  else{
   int f = *adj[v].begin();
   adj[v].erase(adj[v].begin()); adj[f].erase(adj[f].find(v));
   st.push(f);
  }
 }
}
```

18 FFT

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

typedef long long LL;
const double PI = acos(-1.0L);
namespace FFT {
 int N;
 vector<int> perm;
 vector<CD> wp[2];
 void precalculate(int n) {
  assert((n & (n-1)) == 0);
  N = n;
  perm = vector<int> (N, 0);
  for (int k=1; k<N; k<<=1) {
   for (int i=0; i<k; i++) {
    perm[i] <<= 1;
    perm[i+k] = 1 + perm[i];
   }
  }
  wp[0] = wp[1] = vector<CD>(N);
  for (int i=0; i<N; i++) {
   wp[0][i] = CD( cos(2*PI*i/N), sin(2*PI*i/N) );
   wp[1][i] = CD( cos(2*PI*i/N), -sin(2*PI*i/N) );
  }
 }
 void fft(vector<CD> &v, bool invert = false) {
  if (v.size() != perm.size()) precalculate(v.size());
  for (int i=0; i<N; i++)
   if (i < perm[i])
    swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
   for (int i=0, d = N/len; i<N; i+=len) {
    for (int j=0, idx=0; j<len/2; j++, idx += d) {
     CD x = v[i+j];
     CD y = wp[invert][idx]*v[i+j+len/2];
     v[i+j] = x+y;
     v[i+j+len/2] = x-y;
    }
   }
  }
  if (invert) {
   for (int i=0; i<N; i++) v[i]/=N;
  }
 }
 void pairfft(vector<CD> &a, vector<CD> &b, bool invert =
      false) {
  int N = a.size();
  vector<CD> p(N);
  for (int i=0; i<N; i++) p[i] = a[i] + b[i] * CD(0, 1);
  fft(p, invert);
  p.push_back(p[0]);
  for (int i=0; i<N; i++) {
   if (invert) {
    a[i] = CD(p[i].real(), 0);
    b[i] = CD(p[i].imag(), 0);
   }
   else {
    a[i] = (p[i]+conj(p[N-i]))*CD(0.5, 0);
    b[i] = (p[i]-conj(p[N-i]))*CD(0, -0.5);
   }
  }
 }
 vector<LL> multiply(const vector<LL> &a, const vector<LL> &b)
      {
  int n = 1;
  while (n < a.size()+ b.size()) n<<=1;
  vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  fa.resize(n); fb.resize(n);
// fft(fa); fft(fb);
  pairfft(fa, fb);
  for (int i=0; i<n; i++) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vector<LL> ans(n);
  for (int i=0; i<n; i++) ans[i] = round(fa[i].real());
  return ans;
 }
 const int M = 1e9+7, B = sqrt(M)+1;
 vector<LL> anyMod(const vector<LL> &a, const vector<LL> &b) {
  int n = 1;
  while (n < a.size()+ b.size()) n<<=1;
  vector<CD> al(n), ar(n), bl(n), br(n);
  for (int i=0; i<a.size(); i++) al[i] = a[i]%M/B, ar[i] =
       a[i]%M%B;
  for (int i=0; i<b.size(); i++) bl[i] = b[i]%M/B, br[i] =
       b[i]%M%B;
  pairfft(al, ar); pairfft(bl, br);
// fft(al); fft(ar); fft(bl); fft(br);
  for (int i=0; i<n; i++) {
   CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
   CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
   al[i] = ll; ar[i] = lr;
   bl[i] = rl; br[i] = rr;
  }
  pairfft(al, ar, true); pairfft(bl, br, true);
// fft(al, true); fft(ar, true); fft(bl, true); fft(br, true);
  vector<LL> ans(n);
  for (int i=0; i<n; i++) {
   LL right = round(br[i].real()), left = round(al[i].real());;
   LL mid = round(round(bl[i].real()) + round(ar[i].real()));
   ans[i] = ((left%M)*B*B + (mid%M)*B + right)%M;
  }
  return ans;
 }
}
```

19 FWHT

```
const int inv2 = (mod + 1) >> 1;
const int M = (1 << 20);
const int OR = 0;
const int AND = 1;
const int XOR = 2;
struct FWHT{
 int P1[M], P2[M];
 void wt(int *a, int n, int flag = XOR) {
  if (n == 0)
   return;
  int m = n / 2;
  wt(a, m, flag);
  wt(a + m, m, flag);
  for (int i = 0; i < m; i++) {
   int x = a[i], y = a[i + m];
   if (flag == OR)
    a[i] = x, a[i + m] = (x + y) % mod;
   if (flag == AND)
    a[i] = (x + y) % mod, a[i + m] = y;
   if (flag == XOR)
    a[i] = (x + y) % mod, a[i + m] = (x - y + mod) % mod;
  }
 }
 void iwt(int* a, int n, int flag = XOR) {
  if (n == 0)
   return;
  int m = n / 2;
  iwt(a, m, flag);
  iwt(a + m, m, flag);
  for (int i = 0; i < m; i++) {
   int x = a[i], y = a[i + m];
   if (flag == OR)
    a[i] = x, a[i + m] = (y - x + mod) % mod;
   if (flag == AND)
    a[i] = (x - y + mod) % mod, a[i + m] = y;
   if (flag == XOR)
    a[i] = 1LL * (x + y) * inv2 % mod, a[i + m] = 1LL * (x - y
         + mod) * inv2 % mod; // replace inv2 by >>1 if not
         required
  }
 }
 vector<int> multiply(int n, vector<int> A, vector<int> B, int
      flag = XOR) {
  assert(__builtin_popcount(n) == 1);
  A.resize(n);
  B.resize(n);
  for (int i = 0; i < n; i++)
   P1[i] = A[i];
  for (int i = 0; i < n; i++)
   P2[i] = B[i];
  wt(P1, n, flag);
  wt(P2, n, flag);
  for (int i = 0; i < n; i++)
   P1[i] = 1LL * P1[i] * P2[i] % mod;
  iwt(P1, n, flag);
  return vector<int> (P1, P1 + n);
 }
 vector<int> pow(int n, vector<int> A, long long k, int flag =
       XOR) {
  assert(__builtin_popcount(n) == 1);
  A.resize(n);
  for (int i = 0; i < n; i++)
   P1[i] = A[i];
  wt(P1, n, flag);
  for(int i = 0; i < n; i++)
   P1[i] = POW(P1[i], k);
  iwt(P1, n, flag);
  return vector<int> (P1, P1 + n);
 }
} t;
int32_t main(){
 int n;
 cin >> n;
 vector<int> a(M, 0);
 for(int i = 0; i < n; i++) {
  int k;
  cin >> k;
  a[k]++;
 }
 vector<int> v = t.pow(M, a, n+1, AND);
 int ans = 1;
 for(int i = 1; i < M; i++)
  ans += v[i] > 0;
```

```
 cout << ans << '\n';
 return 0;
}
```

20 Fibonacci Shortcut

```
pair<int, int> fib (int n) {
 if (n == 0) return {0, 1};
 auto p = fib(n >> 1);
 int c = p.first * (2 * p.second - p.first);
 int d = p.first * p.first + p.second * p.second;
 if (n & 1) return {d, c + d};
 else return {c, d};
}
```

21 FloorCeilChange

```
vector<int> where_floor_changes(int n){
 int now=1; vector<int>v;
 while(now<=n) {
  v.push_back(now); now=n/(n/now)+1;
 }
 return v;
}
vector<pair<int,int>> where_ceil_changes(int m){
 vector<pair<int,int>>v; int l=1;
 while(l<=m) {
  if(l==m) { v.push_back({m,m}); break;}
  int cl=(m+l-1)/l;
  int r=(m+cl-2)/(cl-1)-1;
  r=min(r,m); r=max(r,l);
  v.push_back({l,r});
  if(r==m) break;
  l=r+1;
 }
 return v;
}
```

22 Floorsum

```
// floor( (a*i+b)/m ) for 0 <= i <= n-1
ll floor_sum(ll n, ll m, ll a, ll b) {
 ll ans = 0;
 if (a >= m) {
  ans += (n - 1) * n * (a / m) / 2;
  a %= m;
 }
 if (b >= m) {
  ans += n * (b / m);
  b %= m;
 }
 ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
 if (y_max == 0) return ans;
 ans += (n - (x_max + a - 1) / a) * y_max;
 ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
 return ans;
}
```

23 Gauss

```
const double EPS = 1e-9;
const int INF = 2;
int gauss (vector < vector<double> > a, vector<double> & ans) {
 int n = (int) a.size();
 int m = (int) a[0].size() - 1;
 vector<int> where (m, -1);
 for (int col=0, row=0; col<m && row<n; ++col) {
  int sel = row;
  for (int i=row; i<n; ++i)
   if (abs (a[i][col]) > abs (a[sel][col]))
    sel = i;
  if (abs (a[sel][col]) < EPS)
   continue;
  for (int i=col; i<=m; ++i)
   swap (a[sel][i], a[row][i]);
  where[col] = row;

  for (int i=0; i<n; ++i)
   if (i != row) {
    double c = a[i][col] / a[row][col];
    for (int j=col; j<=m; ++j)
     a[i][j] -= a[row][j] * c;
   }
  ++row;
 }
 ans.assign (m, 0);
 for (int i=0; i<m; ++i)
  if (where[i] != -1)
   ans[i] = a[where[i]][m] / a[where[i]][i];
 for (int i=0; i<n; ++i) {
  double sum = 0;
  for (int j=0; j<m; ++j)
   sum += ans[j] * a[i][j];
  if (abs (sum - a[i][m]) > EPS)
   return 0;
 }
 for (int i=0; i<m; ++i)
  if (where[i] == -1)
   return INF;
 return 1;
}
```

24 Geo 2D

```
const double pi = 4 * atan(1);const double eps = 1e-6;
inline int dcmp (double x) { if (fabs(x) < eps) return 0; else
     return x < 0 ? -1 : 1; }
double fix_acute(double th) {return th<-pi ? (th+2*pi): th>pi
     ? (th-2*pi) : th;}
inline double getDistance (double x, double y) { return sqrt(x
     * x + y * y); }
inline double torad(double deg) { return deg / 180 * pi; }
struct Pt {
 double x, y;
 Pt (double x = 0, double y = 0): x(x), y(y) {}
 void read () { scanf("%lf%lf", &x, &y); }
 void write () { printf("%lf %lf", x, y); }
 bool operator == (const Pt& u) const { return dcmp(x - u.x)
      == 0 && dcmp(y - u.y) == 0; }
 bool operator != (const Pt& u) const { return !(*this == u); }
 bool operator < (const Pt& u) const { return dcmp(x - u.x) <
      0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }
 bool operator > (const Pt& u) const { return u < *this; }
 bool operator <= (const Pt& u) const { return *this < u ||
      *this == u; }
 bool operator >= (const Pt& u) const { return *this > u ||
      *this == u; }
 Pt operator + (const Pt& u) { return Pt(x + u.x, y + u.y); }
 Pt operator - (const Pt& u) { return Pt(x - u.x, y - u.y); }
 Pt operator * (const double u) { return Pt(x * u, y * u); }
 Pt operator / (const double u) { return Pt(x / u, y / u); }
 double operator * (const Pt& u) { return x*u.y - y*u.x; }
};
typedef Pt Vector;
typedef vector<Pt> Polygon;
struct Line {
 double a, b, c;
 Line (double a = 0, double b = 0, double c = 0): a(a), b(b),
      c(c) {}
};
struct Segment{
 Pt a;Pt b;
 Segment(){}
 Segment(Pt aa,Pt bb) {a=aa,b=bb;}
};
struct DirLine {
 Pt p;Vector v;
 double ang;
 DirLine () {}
 DirLine (Pt p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x);
      }
 bool operator < (const DirLine& u) const { return ang <
      u.ang; }
};
namespace Punctual {
 double getDistance (Pt a, Pt b) { double x=a.x-b.x,
      y=a.y-b.y; return sqrt(x*x + y*y); }
};
```

```
namespace Vectorial {
 double getDot (Vector a, Vector b) { return a.x * b.x + a.y *
      b.y; }
 double getCross (Vector a, Vector b) { return a.x * b.y - a.y
      * b.x; }
 double getLength (Vector a) { return sqrt(getDot(a, a)); }
 double getPLength (Vector a) { return getDot(a, a); }
 double getAngle (Vector u) { return atan2(u.y, u.x); }
 double getSignedAngle (Vector a, Vector b) {return
      getAngle(b)-getAngle(a);}
 Vector rotate (Vector a, double rad) { return
      Vector(a.x*cos(rad)-a.y*sin(rad),
      a.x*sin(rad)+a.y*cos(rad)); }
 Vector ccw(Vector a, double co, double si) {return
      Vector(a.x*co-a.y*si, a.y*co+a.x*si);}
 Vector cw (Vector a, double co, double si) {return
      Vector(a.x*co+a.y*si, a.y*co-a.x*si);}
 Vector scale(Vector a, double s = 1.0) {return a /
      getLength(a) * s;}
 Vector getNormal (Vector a) { double l = getLength(a); return
      Vector(-a.y/l, a.x/l); }
};
namespace ComplexVector {
 typedef complex<double> Pt;
 typedef Pt Vector;
 double getDot(Vector a, Vector b) { return real(conj(a)*b); }
 double getCross(Vector a, Vector b) { return imag(conj(a)*b);
      }
 Vector rotate(Vector a, double rad) { return a*exp(Pt(0,
      rad)); }
};
namespace Linear {
 using namespace Vectorial;
 Line getLine (double x1, double y1, double x2, double y2) {
      return Line(y2-y1, x1-x2, y1*x2-x1*y2); }
 Line getLine (double a, double b, Pt u) { return Line(a, -b,
      u.y * b - u.x * a); }
 bool getIntersection (Line p, Line q, Pt& o) {
  if (fabs(p.a * q.b - q.a * p.b) < eps)
   return false;
  o.x = (q.c * p.b - p.c * q.b) / (p.a * q.b - q.a * p.b);
  o.y = (q.c * p.a - p.c * q.a) / (p.b * q.a - q.b * p.a);
  return true;
 }
 bool getIntersection (Pt p, Vector v, Pt q, Vector w, Pt& o) {
  if (dcmp(getCross(v, w)) == 0) return false;
  Vector u = p - q;
  double k = getCross(w, u) / getCross(v, w);
  o = p + v * k;
  return true;
 }
 double getDistanceToLine (Pt p, Pt a, Pt b) { return
      fabs(getCross(b-a, p-a) / getLength(b-a)); }
 double getDistanceToSegment (Pt p, Pt a, Pt b) {
  if (a == b) return getLength(p-a);
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);
  else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);
  else return fabs(getCross(v1, v2) / getLength(v1));
 }
 double getDistanceSegToSeg (Pt a,Pt b,Pt c,Pt d){
  double Ans=INT_MAX;
  Ans=min(Ans,getDistanceToSegment(a,c,d));
  Ans=min(Ans,getDistanceToSegment(b,c,d));
  Ans=min(Ans,getDistanceToSegment(c,a,b));
  Ans=min(Ans,getDistanceToSegment(d,a,b));
  return Ans;
 }
 Pt getPtToLine (Pt p, Pt a, Pt b) { Vector v = b-a; return
      a+v*(getDot(v, p-a) / getDot(v,v)); }
 bool onSegment (Pt p, Pt a, Pt b) { return dcmp(getCross(a-p,
      b-p)) == 0 && dcmp(getDot(a-p, b-p)) <= 0; }
 bool haveIntersection (Pt a1, Pt a2, Pt b1, Pt b2) {
  if(onSegment(a1,b1,b2)) return true;
  if(onSegment(a2,b1,b2)) return true;
  if(onSegment(b1,a1,a2)) return true;
  if(onSegment(b2,a1,a2)) return true; //Case of touch
```

```
  double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1),
       c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);
  return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
 }
 bool onLeft(DirLine l, Pt p) { return dcmp(l.v * (p-l.p)) >=
      0; }
}
namespace Triangular {
 using namespace Vectorial;
 double getAngle (double a, double b, double c) { return
      acos((a*a+b*b-c*c) / (2*a*b)); }
 double getArea (double a, double b, double c) { double s
      =(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }
 double getArea (double a, double h) { return a * h / 2; }
 double getArea (Pt a, Pt b, Pt c) { return fabs(getCross(b -
      a, c - a)) / 2; }
 double getDirArea (Pt a, Pt b, Pt c) { return getCross(b - a,
      c - a) / 2;}
 //ma/mb/mc = length of median from side a/b/c
 double getArea_(double ma,double mb,double mc) {double
      s=(ma+mb+mc)/2; return 4/3.0 *
      sqrt(s*(s-ma)*(s-mb)*(s-mc));}
 //ha/hb/hc = length of perpendicular from side a/b/c
 double get_Area(double ha,double hb,double hc){
  double H=(1/ha+1/hb+1/hc)/2; double _A_ = 4 * sqrt(H *
       (H-1/ha)*(H-1/hb)*(H-1/hc)); return 1.0/_A_;
 }
 bool PtInTriangle(Pt a, Pt b, Pt c, Pt p){
  double s1 = getArea(a,b,c);
  double s2 = getArea(p,b,c) + getArea(p,a,b) + getArea(p,c,a);
  return dcmp(s1 - s2) == 0;
 }
};
namespace Polygonal {
 using namespace Vectorial;
 using namespace Linear;
 using namespace Triangular;
 double getSignedArea (Pt* p, int n) {
  double ret = 0;
  for (int i = 0; i < n-1; i++)
   ret += (p[i]-p[0]) * (p[i+1]-p[0]);
  return ret/2;
 }
 int getConvexHull (Pt* p, int n, Pt* ch) {
  sort(p, p + n);
  // preparing lower hull
  int m = 0;
  for (int i = 0; i < n; i++){
   while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2],
        p[i]-ch[m-1])) <= 0) m--;
   ch[m++] = p[i];
  }
  // preparing upper hull
  int k = m;
  for (int i = n-2; i >= 0; i--){
   while (m > k && dcmp(getCross(ch[m-1]-ch[m-2],
        p[i]-ch[m-2])) <= 0) m--;
   ch[m++] = p[i];
  }
  if (n > 1) m--;
  return m;
 }
 int isPtInPolygon (Pt o, Pt* p, int n) {
  int wn = 0;
  for (int i = 0; i < n; i++) {
   int j = (i + 1) % n;
   if (onSegment(o, p[i], p[j]) || o == p[i]) return 0;
   int k = dcmp(getCross(p[j] - p[i], o-p[i]));
   int d1 = dcmp(p[i].y - o.y);
   int d2 = dcmp(p[j].y - o.y);
   if (k > 0 && d1 <= 0 && d2 > 0) wn++;
   if (k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  return wn ? -1 : 1;
 }
 void rotatingCalipers(Pt *p, int n, vector<Segment>& sol) {
  sol.clear();
  int j = 1; p[n] = p[0];
  for (int i = 0; i < n; i++) {
   while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) >
        getCross(p[j]-p[i+1], p[i]-p[i+1]))
    j = (j+1) % n;
   sol.push_back(Segment(p[i],p[j]));
   sol.push_back(Segment(p[i + 1],p[j + 1]));
  }
 }
 void rotatingCalipersGetRectangle (Pt *p, int n, double&
      area, double& perimeter) {
  p[n] = p[0];
  int l = 1, r = 1, j = 1;
  area = perimeter = 1e20;
  for (int i = 0; i < n; i++) {
   Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);
   while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v,
        p[(r+1)%n]-p[i])) < 0) r++;
   while (j < r || dcmp(getCross(v, p[j%n]-p[i]) -
        getCross(v,p[(j+1)%n]-p[i])) < 0) j++;
   while (l < j || dcmp(getDot(v, p[l%n]-p[i]) - getDot(v,
        p[(l+1)%n]-p[i])) > 0) l++;
   double w = getDot(v, p[r%n]-p[i])-getDot(v, p[l%n]-p[i]);
   double h = getDistanceToLine (p[j%n], p[i], p[i+1]);
   area = min(area, w * h);
   perimeter = min(perimeter, 2 * w + 2 * h);
  }
 }
 Polygon cutPolygon (Polygon u, Pt a, Pt b) {
  Polygon ret;
  int n = u.size();
  for (int i = 0; i < n; i++) {
   Pt c = u[i], d = u[(i+1)%n];
   if (dcmp((b-a)*(c-a)) >= 0) ret.push_back(c);
   if (dcmp((b-a)*(d-c)) != 0) {
    Pt t;
    getIntersection(a, b-a, c, d-c, t);
    if (onSegment(t, c, d))
     ret.push_back(t);
   }
  }
  return ret;
 }
 int halfPlaneIntersection(DirLine* li, int n, Pt* poly) {
  sort(li, li + n);
  int first, last;
  Pt* p = new Pt[n];
  DirLine* q = new DirLine[n];
  q[first=last=0] = li[0];
  for (int i = 1; i < n; i++) {
   while (first < last && !onLeft(li[i], p[last-1])) last--;
   while (first < last && !onLeft(li[i], p[first])) first++;
   q[++last] = li[i];
   if (dcmp(q[last].v * q[last-1].v) == 0) {
    last--;
    if (onLeft(q[last], li[i].p)) q[last] = li[i];
   }
   if (first < last)
    getIntersection(q[last-1].p, q[last-1].v, q[last].p,
         q[last].v, p[last-1]);
  }
  while (first < last && !onLeft(q[first], p[last-1])) last--;
  if (last - first <= 1) { delete [] p; delete [] q; return 0;
       }
  getIntersection(q[last].p, q[last].v, q[first].p,
      q[first].v, p[last]);
  int m = 0;
  for (int i = first; i <= last; i++) poly[m++] = p[i];
  delete [] p; delete [] q;
  return m;
 }
 Polygon simplify (const Polygon& poly) {
  Polygon ret;
  int n = poly.size();
  for (int i = 0; i < n; i++) {
   Pt a = poly[i];
   Pt b = poly[(i+1)%n];
   Pt c = poly[(i+2)%n];
   if (dcmp((b-a)*(c-b)) != 0 && (ret.size() == 0 || b !=
        ret[ret.size()-1]))
    ret.push_back(b);
  }
  return ret;
 }
 Pt ComputeCentroid( Pt* p,int n){
  Pt c(0,0);
  double scale = 6.0 * getSignedArea(p,n);
  for (int i = 0; i < n; i++){
   int j = (i+1) % n;
   c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
 }
 // pt must be in ccw order with no three collinear Pts
 // returns inside = 1, on = 0, outside = -1
 int PtInConvexPolygon(Pt* pt, int n, Pt p){
  assert(n >= 3);
  int lo = 1 , hi = n - 1 ;
  while(hi - lo > 1){
   int mid = (lo + hi) / 2;
   if(getCross(pt[mid] - pt[0], p - pt[0]) > 0) lo = mid;
   else hi = mid;
  }
  bool in = PtInTriangle(pt[0], pt[lo], pt[hi], p);
  if(!in) return -1;
  if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0) return 0;
  if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return 0;
  if(getCross(pt[hi] - pt[(hi+1)%n], p - pt[(hi+1)%n]) == 0)
       return 0;
  return 1;
 }
 // Calculate [ACW, CW] tangent pair from an external Pt
 #define CW   -1
 #define ACW   1
 int direction(Pt st, Pt ed, Pt q) {return dcmp(getCross(ed -
      st, q - ed));}
 bool isGood(Pt u, Pt v, Pt Q, int dir) {return direction(Q,
      u, v) != -dir;}
 Pt better(Pt u, Pt v, Pt Q, int dir) {return direction(Q, u,
      v) == dir ? u : v;}
 Pt tangents(Pt* pt, Pt Q, int dir, int lo, int hi){
  while(hi - lo > 1){
   int mid = (lo + hi)/2;
   bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
   bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);
   if(pvs && nxt) return pt[mid];
   if(!(pvs || nxt)){
    Pt p1 = tangents(pt, Q, dir, mid+1, hi);
    Pt p2 = tangents(pt, Q, dir, lo, mid - 1);
    return better(p1, p2, Q, dir);
   }
   if(!pvs){
    if(direction(Q, pt[mid], pt[lo]) == dir) hi = mid - 1;
    else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid
         - 1;
    else lo = mid + 1;
   }
   if(!nxt){
    if(direction(Q, pt[mid], pt[lo]) == dir) lo = mid + 1;
    else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid
         - 1;
    else lo = mid + 1;
   }
  }
  Pt ret = pt[lo];
  for(int i = lo + 1; i <= hi; i++) ret = better(ret, pt[i],
       Q, dir);
  return ret;
 }
 // [ACW, CW] Tangent
 pair<Pt, Pt> get_tangents(Pt* pt, int n, Pt Q){
  Pt acw_tan = tangents(pt, Q, ACW, 0, n - 1);
  Pt cw_tan = tangents(pt, Q, CW, 0, n - 1);
  return make_pair(acw_tan, cw_tan);
 }
};
struct Circle {
```

```
 Pt o;double r;
 Circle () {}
 Circle (Pt o, double r = 0): o(o), r(r) {}
 void read () { o.read(), scanf("%lf", &r); }
 Pt pt(double rad) { return Pt(o.x + cos(rad)*r, o.y +
      sin(rad)*r); }
 double getArea (double rad) { return rad * r * r / 2; }
 //area of the circular sector cut by a chord with central
      angle alpha
 double sector(double alpha) {return r * r * 0.5 * (alpha -
      sin(alpha));}
};
namespace Circular {
 using namespace Linear;
 using namespace Vectorial;
 using namespace Triangular;
 int getLineCircleIntersection (Pt p, Pt q, Circle O, double&
      t1, double& t2, vector<Pt>& sol) {
  Vector v = q - p;
  //sol.clear();
  double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;
  double e = a*a+c*c, f = 2*(a*b+c*d), g = b*b+d*d-O.r*O.r;
  double delta = f*f - 4*e*g;
  if (dcmp(delta) < 0) return 0;
  if (dcmp(delta) == 0) {
   t1 = t2 = -f / (2 * e);
   sol.push_back(p + v * t1);
   return 1;
  }
  t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(p + v * t1);
  t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(p + v * t2);
  return 2;
 }
 // signed area of intersection of circle(c.o, c.r) and
 // triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]
 double areaCircleTriIntersection(Circle c, Segment s){
  using namespace Linear;
  double OA = getLength(c.o - s.a);
  double OB = getLength(c.o - s.b);
  // sector
  if (dcmp(getDistanceToSegment(c.o, s.a, s.b) - c.r) >= 0)
   return fix_acute(getSignedAngle(s.a - c.o, s.b - c.o)) *
        (c.r*c.r) / 2.0;
  // triangle
  if (dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
   return getCross(c.o-s.b,s.a-s.b) / 2.0;
  // three part: (A, a) (a, b) (b, B)
  vector<Pt>Sect; double t1,t2;
  getLineCircleIntersection(s.a, s.b, c, t1, t2, Sect);
  return areaCircleTriIntersection(c, Segment(s.a, Sect[0]))
   + areaCircleTriIntersection(c, Segment(Sect[0], Sect[1]))
   + areaCircleTriIntersection(c, Segment(Sect[1], s.b));
 }
 // area of intersecion of circle(c.o, c.r) and simple
      polyson(p[])
 double areaCirclePolygon(Circle c, Polygon p){
  double res = .0;
  int n = p.size();
  for (int i = 0; i < n; ++ i)
   res += areaCircleTriIntersection(c, Segment(p[i],
        p[(i+1)%n]));
  return fabs(res);
 }
 // interior   (d < R - r)  ----> -2
 // interior tangents (d = R - r) ----> -1
 // concentric (d = 0)
 // secants    (R - r < d < R + r) ----> 0
 // exterior tangents (d = R + r) ----> 1
 // exterior   (d > R + r)  ----> 2
 int getPos(Circle o1, Circle o2) {
  using namespace Vectorial;
  double d = getLength(o1.o - o2.o);
  int in = dcmp(d - fabs(o1.r - o2.r)), ex = dcmp(d - (o1.r +
       o2.r));
  return in<0 ? -2 : in==0? -1 : ex==0 ? 1 : ex>0? 2 : 0;
 }
 int getCircleCircleIntersection (Circle o1, Circle o2,
      vector<Pt>& sol) {
  double d = getLength(o1.o - o2.o);
  if (dcmp(d) == 0) {
   if (dcmp(o1.r - o2.r) == 0) return -1;
   return 0;
  }
  if (dcmp(o1.r + o2.r - d) < 0) return 0;
  if (dcmp(fabs(o1.r-o2.r) - d) > 0) return 0;
  Vector v = o2.o - o1.o;
  double co = (o1.r*o1.r + getPLength(v) - o2.r*o2.r) / (2 *
       o1.r * getLength(v));
  double si = sqrt(fabs(1.0 - co*co));
  Pt p1 = scale(cw(v,co, si), o1.r) + o1.o;
  Pt p2 = scale(ccw(v,co, si), o1.r) + o1.o;
  sol.push_back(p1);
  if (p1 == p2) return 1;
  sol.push_back(p2);
  return 2;
 }
 double areaCircleCircle(Circle o1, Circle o2){
  Vector AB = o2.o - o1.o;double d = getLength(AB);
  if(d >= o1.r + o2.r) return 0;
  if(d + o1.r <= o2.r) return pi * o1.r * o1.r;
  if(d + o2.r <= o1.r) return pi * o2.r * o2.r;
  double alpha1 = acos((o1.r * o1.r + d * d - o2.r * o2.r) /
       (2.0 * o1.r * d));
  double alpha2 = acos((o2.r * o2.r + d * d - o1.r * o1.r) /
       (2.0 * o2.r * d));
  return o1.sector(2*alpha1) + o2.sector(2*alpha2);
 }
 int getTangents (Pt p, Circle o, Vector* v) {
  Vector u = o.o - p;
  double d = getLength(u);
  if (d < o.r) return 0;
  else if (dcmp(d - o.r) == 0) {
   v[0] = rotate(u, pi / 2);
   return 1;
  } else {
   double ang = asin(o.r / d);
   v[0] = rotate(u, -ang);
   v[1] = rotate(u, ang);
   return 2;
  }
 }
 int getTangentPts (Pt p, Circle o, vector<Pt>& v) {
  Vector u = p - o.o ;double d = getLength(u);
  if (d < o.r) return 0;
  else if (dcmp(d - o.r) == 0) {
   v.push_back(o.o+u);
   return 1;
  } else {
   double ang = acos(o.r / d);
   u = u / getLength(u) * o.r;
   v.push_back(o.o+rotate(u, -ang));
   v.push_back(o.o+rotate(u, ang));
   return 2;
  }
 }
 int getTangents (Circle o1, Circle o2, Pt* a, Pt* b) {
  int cnt = 0;
  if (dcmp(o1.r-o2.r) < 0) { swap(o1, o2); swap(a, b); }
  double d2 = getPLength(o1.o - o2.o);
  double rdif = o1.r - o2.r, rsum = o1.r + o2.r;
  if (dcmp(d2 - rdif * rdif) < 0) return 0;
  if (dcmp(d2) == 0 && dcmp(o1.r - o2.r) == 0) return -1;
  double base = getAngle(o2.o - o1.o);
  if (dcmp(d2 - rdif * rdif) == 0) {
   a[cnt] = o1.pt(base); b[cnt] = o2.pt(base); cnt++;
   return cnt;
  }
  double ang = acos( (o1.r - o2.r) / sqrt(d2) );
  a[cnt] = o1.pt(base+ang); b[cnt] = o2.pt(base+ang); cnt++;
  a[cnt] = o1.pt(base-ang); b[cnt] = o2.pt(base-ang); cnt++;
  if (dcmp(d2 - rsum * rsum) == 0) {
   a[cnt] = o1.pt(base); b[cnt] = o2.pt(pi+base); cnt++;
  }
  else if (dcmp(d2 - rsum * rsum) > 0) {
   double ang = acos( (o1.r + o2.r) / sqrt(d2) );
   a[cnt] = o1.pt(base+ang); b[cnt] = o2.pt(pi+base+ang);
        cnt++;
   a[cnt] = o1.pt(base-ang); b[cnt] = o2.pt(pi+base-ang);
        cnt++;
  }
  return cnt;
 }
 Circle CircumscribedCircle(Pt p1, Pt p2, Pt p3) {
  double Bx = p2.x - p1.x, By = p2.y - p1.y;
  double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
  double D = 2 * (Bx * Cy - By * Cx);
  double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy *
       Cy)) / D + p1.x;
  double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By *
       By)) / D + p1.y;
  Pt p = Pt(cx, cy);
  return Circle(p, getLength(p1 - p));
 }
 Circle InscribedCircle(Pt p1, Pt p2, Pt p3) {
  double a = getLength(p2 - p3);double b = getLength(p3 - p1);
  double c = getLength(p1 - p2);
  Pt p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
  return Circle(p, getDistanceToLine(p, p1, p2));
 }
 //distance From P : distance from Q = rp : rq
 Circle getApolloniusCircle(const Pt& P,const Pt& Q, double
      rp, double rq ){
  rq *= rq ;rp *= rp ;
  double a = rq - rp ;
  assert(dcmp(a));
  double g = rq * P.x - rp * Q.x ; g /= a ;
  double h = rq * P.y - rp * Q.y ; h /= a ;
  double c = rq*P.x*P.x-rp*Q.x*Q.x+rq*P.y*P.y-rp*Q.y*Q.y ;
  c /= a ;
  Pt o(g,h);
  double R = g*g +h*h - c ;
  R = sqrt(R);
  return Circle(o,R);
 }
};
//Polar Sort
inline bool up (Pt p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (Pt a, Pt b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) <
       up(b);
});
```

25 Geo 3D

```
const double pi = 4 * atan(1);
const double eps = 1e-10;
inline int dcmp (double x) { if (fabs(x) < eps) return 0; else
     return x < 0 ? -1 : 1; }
inline double torad(double deg) { return deg / 180 * pi; }
struct Point{
 double x, y;
 Point (double x = 0, double y = 0): x(x), y(y) {}
 Point operator + (const Point& u) { return Point(x + u.x, y +
      u.y); }
 Point operator - (const Point& u) { return Point(x - u.x, y -
      u.y); }
 Point operator * (const double u) { return Point(x * u, y *
      u); }
 Point operator / (const double u) { return Point(x / u, y /
      u); }
 double operator * (const Point& u) { return x*u.y - y*u.x; }
};
struct Pt3D{
 double x, y, z;
 Pt3D() {}
 void read () {cin>>x>>y>>z;}
 void write () {cout<<x<<" --- "<<y<<" --- "<<z<<"\n";}
 Pt3D(double x, double y, double z) : x(x), y(y), z(z) {}
 Pt3D(const Pt3D &p) : x(p.x), y(p.y), z(p.z) {}
 Pt3D operator +(Pt3D b) {return Pt3D(x+b.x,y+b.y, z+b.z);}
 Pt3D operator -(Pt3D b) {return Pt3D(x-b.x,y-b.y, z-b.z);}
```

```
 Pt3D operator *(double b) {return Pt3D(x*b,y*b, z*b);}
 Pt3D operator /(double b) {return Pt3D(x/b,y/b, z/b);}
 bool operator <(Pt3D b) {return make_pair(make_pair(x,y),z) <
      make_pair(make_pair(b.x,b.y),b.z);}
 bool operator ==(Pt3D b) {return dcmp(x-b.x)==0 &&
      dcmp(y-b.y) == 0 && dcmp(z-b.z) == 0;}
};
typedef Pt3D Vector3D;
typedef vector<Point> Polygon;
typedef vector<Pt3D> Polyhedron;
namespace Vectorial{
 double getDot (Vector3D a, Vector3D b) {return
      a.x*b.x+a.y*b.y+a.z*b.z;}
 Vector3D getCross(Vector3D a, Vector3D b) {return
      Pt3D(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x);}
 double getLength (Vector3D a) {return sqrt(getDot(a, a)); }
 double getPLength (Vector3D a) {return getDot(a, a); }
 Vector3D unitVector(Vector3D v) {return v/getLength(v);}
 double getUnsignedAngle(Vector3D u,Vector3D v){
  double cosTheta = getDot(u,v)/getLength(u)/getLength(v);
  cosTheta = max(-1.0,min(1.0,cosTheta));
  return acos(cosTheta);
 }
 Vector3D rotate(Vector3D v,Vector3D a,double rad){
  a = unitVector(a);
  return v * cos(rad) + a * (1 - cos(rad)) * getDot(a,v) +
       getCross(a,v) * sin(rad);
 }
}
struct Line3D{
 Vector3D v; Pt3D o;
 Line3D() {};
 Line3D(Vector3D v,Pt3D o):v(v),o(o){}
 Pt3D getPoint(double t) {return o + v*t;}
};
namespace Linear{
 using namespace Vectorial;
 double getDistSq(Line3D l, Pt3D p) {return
      getPLength(getCross(l.v,p-l.o))/getPLength(l.v);}
 double getDistLinePoint(Line3D l, Pt3D p) {return
      sqrt(getDistSq(l,p));}
 bool cmp(Line3D l,Pt3D p, Pt3D q) {return getDot(l.v,p) <
      getDot(l.v,q);}
 Pt3D projection(Line3D l,Pt3D p) {return l.o + l.v *
      getDot(l.v,p-l.o)/getPLength(l.v);}
 Pt3D reflection(Line3D l,Pt3D p) {return
      projection(l,p)+projection(l,p)-p;}
 double getAngle(Line3D l,Line3D m) {return
      getUnsignedAngle(l.v,m.v);}
 bool isParallel(Line3D p,Line3D q) {return
      dcmp(getPLength(getCross(p.v,q.v))) == 0;}
 bool isPerpendicular(Line3D p,Line3D q) {return
      dcmp(getDot(p.v,q.v)) == 0;}
 double getDist(Line3D l, Line3D m){
  Vector3D n = getCross(l.v, m.v);
  if(getPLength(n) == 0) return getDistLinePoint(l,m.o);
  else return fabs(getDot(m.o-l.o , n)) / getLength(n);
 }
 Pt3D getClosestPointOnLine1(Line3D l,Line3D m){
  Vector3D n = getCross(l.v, m.v);
  Vector3D n2 = getCross(m.v, n);
  return l.o + l.v * getDot(m.o-l.o, n2) / getDot(l.v, n2);
 }
}
struct Plane{
 Vector3D n; //normal n
 double d; //getDot(n,p) = d for any point p on the plane
 Plane() {}
 Plane(Vector3D n, double d) : n(n), d(d) {}
 Plane(Vector3D n, Pt3D p) : n(n), d(Vectorial :: getDot(n,p))
      {}
 Plane(const Plane &p) : n(p.n), d(p.d) {}
};
namespace Planar{
 using namespace Vectorial;
 Plane getPlane(Pt3D a,Pt3D b,Pt3D c) {return
      Plane(getCross(b-a,c-a),a);}
 Plane translate(Plane p,Vector3D t) {return Plane(p.n,
      p.d+getDot(p.n,t));}
 Plane shiftUp(Plane p,double dist) {return Plane(p.n,
      p.d+dist*getLength(p.n));}
 Plane shiftDown(Plane p,double dist) {return Plane(p.n,
      p.d-dist*getLength(p.n));}
 double getSide(Plane p,Pt3D a) {return getDot(p.n,a)-p.d;}
 double getDistance(Plane p,Pt3D a) {return
      fabs(getSide(p,a))/getLength(p.n);}
 Pt3D projection(Plane p,Pt3D a) {return
      a-p.n*getSide(p,a)/getPLength(p.n);}
 Pt3D reflection(Plane p,Pt3D a) {return
      a-p.n*getSide(p,a)/getPLength(p.n)*2;}
 bool intersect(Plane p, Line3D l, Pt3D& a){
  if(dcmp(getDot(p.n,l.v)) == 0) return false;
  a = l.o - l.v * getSide(p,l.o) / getDot(p.n,l.v);
  return true;
 }
 bool intersect(Plane p,Plane q,Line3D& l){
  l.v = getCross(p.n,q.n);
  if(dcmp(getPLength(l.v)) == 0) return false;
  l.o = getCross(q.n*p.d - p.n*q.d , l.v) / getPLength(l.v);
  return true;
 }
 double getAngle(Plane p,Plane q) {return
      getUnsignedAngle(p.n,q.n);}
 bool isParallel(Plane p,Plane q) {return
      dcmp(getPLength(getCross(p.n,q.n))) == 0;}
 bool isPerpendicular(Plane p,Plane q) {return
      dcmp(getDot(p.n,q.n)) == 0;}
 bool getAngle(Plane p,Line3D l) {return pi/2.0 -
      getUnsignedAngle(p.n,l.v);}
 bool isParallel(Plane p,Line3D l) {return
      dcmp(getDot(p.n,l.v)) == 0;}
 bool isPerpendicular(Plane p,Line3D l) {return
      dcmp(getPLength(getCross(p.n,l.v))) == 0;}
 Line3D perpThrough(Plane p,Pt3D a) {return Line3D(p.n,a);}
 Plane perpThrough(Line3D l,Pt3D a) {return Plane(l.v,a);}
 //Modify p.n if necessary with respect to the reference point
 Vector3D rotateCCW90(Plane p,Vector3D d) {return
      getCross(p.n,d);}
 Vector3D rotateCW90(Plane p,Vector3D d) {return
      getCross(d,p.n);}
 pair<Pt3D, Pt3D> TwoPointsOnPlane(Plane p){
  Vector3D N = p.n; double D = p.d;
  assert(dcmp(N.x) != 0 || dcmp(N.y) != 0 || dcmp(N.z) != 0);
  if(dcmp(N.x) == 0 && dcmp(N.y) == 0) return
       {Pt3D(1,0,D/N.z), Pt3D(0,1,D/N.z)};
  if(dcmp(N.y) == 0 && dcmp(N.z) == 0) return
       {Pt3D(D/N.x,1,0), Pt3D(D/N.x,0,1)};
  if(dcmp(N.z) == 0 && dcmp(N.x) == 0) return
       {Pt3D(1,D/N.y,0), Pt3D(0,D/N.y,1)};
  if(dcmp(N.x) == 0) return {Pt3D(1,D/N.y,0), Pt3D(0,0,D/N.z)};
  if(dcmp(N.y) == 0) return {Pt3D(0,1,D/N.z), Pt3D(D/N.x,0,0)};
  if(dcmp(N.z) == 0) return {Pt3D(D/N.x,0,1), Pt3D(0,D/N.y,0)};
  if(dcmp(D)!=0) return {Pt3D(D/N.x,0,0), Pt3D(0,D/N.y,0)};
  return {Pt3D(N.y,-N.x,0), Pt3D(-N.y,N.x,0)};
 }
 Point From3Dto2D(Plane p, Pt3D a){
  assert( dcmp(getSide(p,a)) == 0 );
  auto Pair = TwoPointsOnPlane(p);
  Pt3D A = Pair.first;
  Pt3D B = Pair.second;
  Vector3D Z = p.n;   Z = Z / getLength(Z);
  Vector3D X = B - A;    X = X / getLength(X);
  Vector3D Y = getCross(Z,X);
  Vector3D v = a - A;
  assert( dcmp(getDot(v,Z)) == 0);
  return Point(getDot(v,X),getDot(v,Y));
 }
 Pt3D From2Dto3D(Plane p, Point a){
  auto Pair = TwoPointsOnPlane(p);
  Pt3D A = Pair.first;
  Pt3D B = Pair.second;
  Vector3D Z = p.n;   Z = Z / getLength(Z);
  Vector3D X = B - A;    X = X / getLength(X);
  Vector3D Y = getCross(Z,X);
  return A + X * a.x + Y * a.y;
 }
}
struct Sphere{
 Pt3D c;
 double r;
 Sphere() {}
 Sphere(Pt3D c, double r) : c(c), r(r) {}
 //Spherical cap with polar angle theta
 double Height(double alpha) {return r*(1-cos(alpha));}
 double BaseRadius(double alpha) {return r*sin(alpha);}
 double Volume(double alpha) {double h = Height(alpha); return
      pi*h*h*(3*r-h)/3.0;}
 double SurfaceArea(double alpha) {double h = Height(alpha);
      return 2*pi*r*h;}
};
namespace Spherical{
 using namespace Vectorial;
 using namespace Planar;
 using namespace Linear;
 Sphere CircumscribedSphere(Pt3D a,Pt3D b,Pt3D c,Pt3D d){
  assert( dcmp(getSide(getPlane(a,b,c), d)) != 0);
  Plane U = Plane(a-b, (a+b)/2);
  Plane V = Plane(b-c, (b+c)/2);
  Plane W = Plane(c-d, (c+d)/2);
  Line3D l1,l2;
  bool ret1 = intersect(U,V,l1);
  bool ret2 = intersect(V,W,l2);
  assert(ret1 == true && ret2 == true);
  assert( dcmp(getDist(l1,l2)) == 0);
  Pt3D C = getClosestPointOnLine1(l1,l2);
  return Sphere(C, getLength(C-a));
 }
 pair<double,double> SphereSphereIntersection(Sphere s1,Sphere
      s2){
  double d = getLength(s1.c-s2.c);
  if(dcmp(d - s1.r -s2.r) >= 0) return {0,0};
  double R1 = max(s1.r,s2.r); double R2 = min(s1.r,s2.r);
  double y = R1 + R2 - d;
  double x = (R1*R1 - R2*R2 + d*d) / (2*d);
  double h1 = R1 - x;
  double h2 = y - h1;
  double Volume = pi*h1*h1*(3*R1-h1)/3.0 +
       pi*h2*h2*(3*R2-h2)/3.0;
  double SurfaceArea = 2*pi*R1*h1 + 2*pi*R2*h2;
  return make_pair(SurfaceArea,Volume);
 }
 Pt3D getPointOnSurface(double r,double Lat,double Lon){
  Lat = torad(Lat); //North-South
  Lon = torad(Lon); //East-West
  return Pt3D(r*cos(Lat)*cos(Lon), r*cos(Lat)*sin(Lon),
       r*sin(Lat));
 }
 int intersect(Sphere s,Line3D l, vector<Pt3D>& ret){
  double h2 = s.r*s.r - getDistSq(l,s.c);
  if(dcmp(h2)<0) return 0;
  Pt3D p = projection(l,s.c);
  if(dcmp(h2) == 0) {ret.push_back(p); return 1;}
  Vector3D h = l.v * sqrt(h2) / getLength(l.v);
  ret.push_back(p-h); ret.push_back(p+h); return 2;
 }
 double GreatCircleDistance(Sphere s,Pt3D a,Pt3D b){
  return s.r * getUnsignedAngle(a-s.c, b-s.c);
 }
}
namespace Poly{
 using namespace Vectorial;
 Sphere SmallestEnclosingSphere(Polyhedron p){
  int n = p.size();
  Pt3D C(0,0,0);
  for(int i=0; i<n; i++) C = C + p[i];
  C = C / n;
  double P = 0.1;
  int pos = 0;
  int Accuracy = 70000;
  for (int i = 0; i < Accuracy; i++) {
   pos = 0;
   for (int j = 1; j < n; j++){
```

```
    if(getPLength(C - p[j]) > getPLength(C - p[pos])) pos = j;
   }
   C = C + (p[pos] - C)*P;
   P *= 0.998;
  }
  return Sphere(C, getPLength(C - p[pos]));
 }
}
```

26 HLD

```
struct HLD{
 vector<int> parent, depth, heavy, head, pos;
 int cur_pos; segtree seg;
 int dfs(int v, vector<vector<int>> const& adj) {
  int size = 1; int max_c_size = 0;
  for (int c : adj[v]) {
   if (c != parent[v]) {
    parent[c] = v, depth[c] = depth[v] + 1;
    int c_size = dfs(c, adj);
    size += c_size;
    if (c_size > max_c_size) max_c_size = c_size, heavy[v] = c;
   }
  }
  return size;
 }
 void decompose(int v, int h, vector<vector<int>> const& adj) {
  head[v] = h, pos[v] = cur_pos++;
  if (heavy[v] != -1)
   decompose(heavy[v], h, adj);
  for (int c : adj[v]) {
   if (c != parent[v] && c != heavy[v])
    decompose(c, c, adj);
  }
 }
 void init(vector<vector<int>> const& adj, vector<ll>&a) {
  int n = adj.size();
  parent = vector<int>(n); depth = vector<int>(n);
  heavy = vector<int>(n, -1); head = vector<int>(n);
  pos = vector<int>(n); cur_pos = 0;
  dfs(0, adj); decompose(0, 0, adj);
  vector<ll>tmp(n);
  for(int i=0; i<n; i++) {
   tmp[pos[i]] = a[i];
  }
  seg.init(n,tmp);
 }
 int query(int a, int b) {
  ll res = 0;
  for (; head[a] != head[b]; b = parent[head[b]]) {
   if (depth[head[a]] > depth[head[b]])
    swap(a, b);
   int cur_heavy_path_max = seg.query(pos[head[b]], pos[b]);
   res += cur_heavy_path_max;
  }
  if (depth[a] > depth[b])
   swap(a, b);
  int last_heavy_path_max = seg.query(pos[a], pos[b]);
  res += last_heavy_path_max;
  return res;
 }
 void update(int a, int b, int x) {
  for (; head[a] != head[b]; b = parent[head[b]]) {
   if (depth[head[a]] > depth[head[b]]) swap(a, b);
   seg.update(pos[head[b]], pos[b], x);
  }
  if (depth[a] > depth[b]) swap(a, b);
  seg.update(pos[a], pos[b],x);
 }
};
```

27 Hackenbush

```
struct hackenbush {
 int n;
 vector<vector<int>> adj;
 hackenbush(int n) : n(n), adj(n) { }
 void add_edge(int u, int v) {
  adj[u].push_back(v);
  if (u != v) adj[v].push_back(u);
 }
 // r is the only root connecting to the ground
 int grundy(int r) {
  vector<int> num(n), low(n);
  int t = 0;
  function<int(int, int)> dfs = [&](int p, int u) {
   num[u] = low[u] = ++t;
   int ans = 0;
   for (int v : adj[u]) {
    if (v == p) { p += 2 * n; continue; }
    if (num[v] == 0) {
     int res = dfs(u, v);
     low[u] = min(low[u], low[v]);
     if (low[v] > num[u]) ans ^= (1 + res) ^ 1; // bridge
     else ans ^= res; // non bridge
    } else low[u] = min(low[u], num[v]);
   }
   if (p > n) p -= 2 * n;
   for (int v : adj[u])
    if (v != p && num[u] <= num[v]) ans ^= 1;
   return ans;
  };
  return dfs(-1, r);
 }
};
int main() {
 int cases; scanf("%d", &cases);
 for (int icase = 0; icase < cases; ++icase) {
  int n; scanf("%d", &n);
  vector<int> ground(n);
  int r;
  for (int i = 0; i < n; ++i) {
   scanf("%d", &ground[i]);
   if (ground[i] == 1) r = i;
  }
  int ans = 0;
  hackenbush g(n);
  for (int i = 0; i < n - 1; ++i) {
   int u, v;
   scanf("%d %d", &u, &v);
   --u; --v;
   if (ground[u]) u = r;
   if (ground[v]) v = r;
   if (u == v) ans ^= 1;
   else g.add_edge(u, v);
  }
  int res = ans ^ g.grundy(r);
  printf("%d\n", res != 0);
 }
}
```

28 Hashing 2D

```
int mods[2] = {1000000007, 1000000009};
int bases[2] = {137, 281};
int pwbase[2][MAX];
void Preprocess(){
 pwbase[0][0] = pwbase[1][0] = 1;
 for(int i = 0; i < 2; i++) {
  for(int j = 1; j < MAX; j++) {
   pwbase[i][j] = (pwbase[i][j - 1] *1ll* bases[i]) % mods[i];
  }
 }
}
struct Hashing{
 int hsh[2][MAX];
 string str;
 void setstr(string &_str) {
  str = _str;
  hsh[0][str.size()] = 0;
  hsh[1][str.size()] = 0;
  Build();
 }
 void Build() {
  for(int i = str.size() - 1; i >= 0; i--) {
   for(int j = 0; j < 2; j++) {
    hsh[j][i] = ((hsh[j][i + 1] *1ll* bases[j] % mods[j]) +
         str[i]);
    if(hsh[j][i]>=mods[j])
        hsh[j][i]-=mods[j];
   }
  }
 }
 pair<int,int> GetHash(int i, int j) {
  assert(i <= j);
  int tmp1 = (hsh[0][i] - (hsh[0][j + 1] *1ll* pwbase[0][j - i
       + 1]) % mods[0]);
  int tmp2 = (hsh[1][i] - (hsh[1][j + 1] *1ll* pwbase[1][j - i
       + 1]) % mods[1]);
  if(tmp1 < 0)
   tmp1 += mods[0];
  if(tmp2 < 0)
   tmp2 += mods[1];
  return make_pair(tmp1, tmp2);
 }
};
```

29 Hopcroft

```
// If input graph is not given in L-R manner, make it so by
     coloring.
// Input graph must be bipartite
const int N=200*200+5;
struct HopcroftKarp
{
 static const int inf = 1e9; int n;
 vector<int> l, r, d; vector<vector<int>> g;
 HopcroftKarp(int _n, int _m) {
  n = _n; int p = _n + _m + 1;
  g.resize(p); l.resize(p, 0); r.resize(p, 0); d.resize(p, 0);
 }
 void add_edge(int u, int v) {
  g[u].push_back(v + n); //right id is increased by n, so is
       l[u]
 }
 bool bfs() {
  queue<int> q;
  for (int u = 1; u <= n; u++) {
   if (!l[u])
    d[u] = 0, q.push(u);
   else
    d[u] = inf;
  }
  d[0] = inf;
  while (!q.empty()) {
   int u = q.front(); q.pop();
   for (auto v : g[u]) { if (d[r[v]] == inf) {
    d[r[v]] = d[u] + 1; q.push(r[v]);
   }
  }
  }
  return d[0] != inf;
 }
 bool dfs(int u) {
  if (!u) return true;
  for (auto v : g[u]){ if(d[r[v]] == d[u] + 1 && dfs(r[v])){
    l[u] = v; r[v] = u;
    return true;
  }
  }
  d[u] = inf;
  return false;
 }
 int maximum_matching(){
  int ans = 0;
  while (bfs()){ for(int u = 1; u <= n; u++)
    if (!l[u] && dfs(u)) ans++;
  }
  return ans;
 }
};
```

30 Hungarian

```
namespace wm{
```

```
bool vis[N]; int U[N],V[N],P[N];
int way[N],minv[N],match[N],ar[N][N];
///n=no of row, m=no of col,1 based,flag=MAXIMIZE/MINIMIZE
///match[i]=the column to which row i is matched
int hungarian(int n,int m,int mat[N][N],int flag){
  clr(U), clr(V), clr(P), clr(ar), clr(way);
  for (int i = 1; i <= n; i++){
    for (int j = 1; j <= m; j++){
      ar[i][j] = mat[i][j];
      if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];
    }
  }
  if (n > m) m = n;
  int i, j, a, b, c, d, r, w;
  for (i = 1; i <= n; i++){
    P[0] = i, b = 0;
    for (j=0; j<=m; j++) minv[j]=inf, vis[j] = 0;
    do{
      vis[b] = true; a = P[b], d = 0, w = inf;
      for (j = 1; j <= m; j++){
        if (!vis[j]){
          r = ar[a][j] - U[a] - V[j];
          if (r < minv[j]) minv[j] = r, way[j]=b;
          if (minv[j] < w) w = minv[j], d = j;
        }
      }
      for (j = 0; j <= m; j++){
        if (vis[j]) U[P[j]] += w, V[j] -= w;
        else minv[j] -= w;
      }
      b = d;
    } while (P[b] != 0);
    do{
      d = way[b]; P[b] = P[d], b = d;
    } while (b != 0);
  }
  for (j = 1; j <= m; j++) match[P[j]] = j;
  return (flag == MINIMIZE) ? -V[0] : V[0];
}
}
```

31 IntersectingSegmentSweepLine

```
struct seg {
 pt p, q;int id;
 double get_y(double x) const {
  if (abs(p.x - q.x) < EPS)return p.y;
  return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
 }
};

bool intersect1d(double l1, double r1, double l2, double r2) {
 if (l1 > r1) swap(l1, r1);
 if (l2 > r2) swap(l2, r2);
 return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
 double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x -
      a.x);
 return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}

bool intersect(const seg& a, const seg& b) {
 return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
     intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
     vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
     vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}

bool operator<(const seg& a, const seg& b){
 double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
 return a.get_y(x) < b.get_y(x) - EPS;
}

struct event {
 double x;int tp, id;
 event() {}
 event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
 bool operator<(const event& e) const {
  if (abs(x - e.x) > EPS)return x < e.x;
  return tp > e.tp;
 }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
 return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
 return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
 int n = (int)a.size();vector<event> e;
 for (int i = 0; i < n; ++i) {
  e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
  e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
 }
 sort(e.begin(), e.end());s.clear();
 where.resize(a.size());
 for (size_t i = 0; i < e.size(); ++i) {
  int id = e[i].id;
  if (e[i].tp == +1) {
   set<seg>::iterator nxt = s.lower_bound(a[id]), prv =
        prev(nxt);
   if (nxt != s.end() && intersect(*nxt, a[id]))
    return make_pair(nxt->id, id);
   if (prv != s.end() && intersect(*prv, a[id]))
    return make_pair(prv->id, id);
   where[id] = s.insert(nxt, a[id]);
  } else {
   set<seg>::iterator nxt = next(where[id]), prv =
        prev(where[id]);
   if (nxt != s.end() && prv != s.end() && intersect(*nxt,
        *prv))
    return make_pair(prv->id, nxt->id);
   s.erase(where[id]);
  }
 }
 return make_pair(-1, -1);
}
```

32 KnuthDP

```
int n,k;int a[N];ll dp[N][N];int opt[N][N];int cost[N][N];
void TEST_CASES(){
 memset(dp,0,sizeof dp);
 for(int i=1;i<=n;i++){opt[0][i] = 0;}
 for(int i=1;i<=k;i++){opt[i][n+1] = n;}
 for(int group = 1;group<=k;group++){
  for(int i=n;i>=1;i--){
   for(int last = opt[group-1][i];last<=
        opt[group][i+1];last++){
    ll val = dp[group-1][last] + cost[last+1][i];
    if(val>dp[group][i]){
     dp[group][i] = val;
   opt[group][i] = last;
    }
   }
  }
 }
 cout<<dp[k][n]<<"\n";
}
```

33 LCA

```
template <class T>
struct RMQ { // 0-based
 vector<vector<T>> rmq;
 T kInf = numeric_limits<T>::max();
 void build(const vector<T>& V) {
  int n = V.size(), on = 1, dep = 1;
  while (on < n) on *= 2, ++dep;
  rmq.assign(dep, V);
  for (int i = 0; i < dep - 1; ++i)
   for (int j = 0; j < n; ++j) {
    rmq[i + 1][j] = min(rmq[i][j], rmq[i][min(n - 1, j + (1 <<
         i))]);
   }
 }
 T query(int a, int b) { // [a, b)
  if (b <= a) return kInf;
  int dep = 31 - __builtin_clz(b - a); // log(b - a)
  return min(rmq[dep][a], rmq[dep][b - (1 << dep)]);
 }
};
struct LCA { // 0-based
 vector<int> enter, depth, exxit;
 vector<vector<int>> G;
 vector<pair<int, int>> linear;
 RMQ<pair<int, int>> rmq;
 int timer = 0;
 LCA() {}
 LCA(int n) : enter(n, -1), exxit(n, -1), depth(n), G(n),
      linear(2 * n) {}
 void dfs(int node, int dep) {
  linear[timer] = {dep, node};
  enter[node] = timer++;
  depth[node] = dep;
  for (auto vec : G[node])
  if (enter[vec] == -1) {
   dfs(vec, dep + 1);
   linear[timer++] = {dep, node};
  }
  exxit[node] = timer;
 }
 void add_edge(int a, int b) {
  G[a].push_back(b);
  G[b].push_back(a);
 }
 void build(int root) {
  dfs(root, 0);
  rmq.build(linear);
 }
 int query(int a, int b) {
  a = enter[a], b = enter[b];
  return rmq.query(min(a, b), max(a, b) + 1).second;
 }
 int dist(int a, int b) {
  return depth[a] + depth[b] - 2 * depth[query(a, b)];
 }
};
```

34 LCABinaryLift

```
int n, l;
vector<vector<int>> adj;
int timer;
vector<int> tin, tout;
vector<vector<int>> up;
void dfs(int v, int p){
 tin[v] = ++timer;
 up[v][0] = p;
 for (int i = 1; i <= l; ++i)
  up[v][i] = up[up[v][i-1]][i-1];
 for (int u : adj[v]) {
  if (u != p)
   dfs(u, v);
 }
 tout[v] = ++timer;
}
bool is_ancestor(int u, int v){
 return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v){
 if (is_ancestor(u, v))
  return u;
 if (is_ancestor(v, u))
  return v;
 for (int i = l; i >= 0; --i) {
  if (!is_ancestor(up[u][i], v))
   u = up[u][i];
 }
 return up[u][0];
}
void preprocess(int root) {
 tin.resize(n);
```

```
 tout.resize(n);
 timer = 0;
 l = ceil(log2(n));
 up.assign(n, vector<int>(l + 1));
 dfs(root, root);
}
```

35 LCT rooted

```
typedef pair< int, int >Linear;
Linear compose(const Linear &p, const Linear &q)
{
 return Linear(mul(p.first, q.first),
      sum(mul(q.second, p.first), p.second));
}
struct SplayTree
{
 struct Node {
  int ch[2] = {0, 0}, p = 0;
  long long self = 0, path = 0;//Path aggregates
  long long sub = 0, vir = 0;//Subtree aggregate
  int size = 1; bool flip = 0;// Lazy tags
  Linear _self{1, 0}, shoja{1, 0}, ulta{1, 0};
 };
 vector<Node> T;
 SplayTree(int n) : T(n + 1) {
  T[0].size = 0;
 }
 void push(int x) {
  if (!x || !T[x].flip)
   return;
  int l = T[x].ch[0], r = T[x].ch[1];
  T[l].flip ^= 1, T[r].flip ^= 1;
  swap(T[x].ch[0], T[x].ch[1]); T[x].flip = 0;
  swap(T[x].shoja, T[x].ulta);
 }
 void pull(int x) {
  int l=T[x].ch[0],r=T[x].ch[1];
  push(l);
  push(r);
  T[x].size = T[l].size + T[r].size + 1;
  T[x].path = T[l].path + T[x].self + T[r].path;
  T[x].sub=T[x].vir+T[l].sub+T[r].sub+T[x].self;
  T[x].shoja = compose(T[r].shoja,
       compose(T[x]._self, T[l].shoja));
  T[x].ulta = compose(T[l].ulta,
       compose(T[x]._self, T[r].ulta));
 }
 void set(int x, int d, int y) {
  T[x].ch[d] = y; T[y].p = x; pull(x);
 }
 void splay(int x) {
  auto dir = [&](int x)
  {
   int p = T[x].p;
   if (!p) return -1;
   return T[p].ch[0]==x?0:T[p].ch[1]==x?1:-1;
  };
  auto rotate = [&](int x)
  {
   int y = T[x].p,z=T[y].p,dx=dir(x),dy=dir(y);
   set(y, dx, T[x].ch[!dx]); set(x, !dx, y);
   if (~dy) set(z, dy, x);
   T[x].p = z;
  };
  for (push(x); ~dir(x); )
  {
   int y = T[x].p,z = T[y].p;
   push(z); push(y); push(x);
   int dx = dir(x), dy = dir(y);
   if (~dy) rotate(dx!=dy?x:y);
   rotate(x);
  }
 }
 int KthNext(int x, int k) {
  assert(k > 0); splay(x);
  x = T[x].ch[1];
  if (T[x].size < k) return -1;
  while (true)
  {
   push(x); int l = T[x].ch[0], r = T[x].ch[1];
   if (T[l].size+1 == k) return x;
   if (k <= T[l].size) x = l;
   else k -= T[l].size+1, x = r;
  }
 }
};
struct LinkCut : SplayTree
{
 LinkCut(int n) : SplayTree(n) {}
 int access(int x) {
  int u = x, v = 0;
  for (; u; v = u, u = T[u].p)
  {
   splay(u); int& ov = T[u].ch[1];
   T[u].vir += T[ov].sub; T[u].vir -= T[v].sub;
   ov = v; pull(u);
  }
  splay(x);
  return v;
 }
 void reroot(int x) {
  access(x); T[x].flip ^= 1; push(x);
 }
///makes v parent of u !(u must be a root)
 void Link(int u, int v) {
  reroot(u); access(v); T[v].vir += T[u].sub;
  T[u].p = v; pull(v);
 }
///removes edge between u and v
 void Cut(int u, int v) {
  int _u = FindRoot(u); reroot(u);
  access(v); T[v].ch[0] = T[u].p = 0;
  pull(v); reroot(_u);
 }
//Rooted tree LCA.Returns 0 if u v not connected
 int LCA(int u, int v) {
  if (u == v) return u;
  access(u); int ret = access(v);
  return T[u].p ? ret : 0;
 }
//Query subtree of u where v is outside the sbtr
 long long Subtree(int u, int v) {
  int _v = FindRoot(v); reroot(v); access(u);
  long long ans = T[u].vir + T[u].self;
  reroot(_v);
  return ans;
 }
 long long Path(int u, int v) {
  int _u = FindRoot(u); reroot(u); access(v);
  long long ans = T[v].path;
  reroot(_u);
  return ans;
 }
 Linear _Path(int u, int v) {
  reroot(u); access(v);
  return T[v].shoja;
 }
 void Update(int u, long long v) {
  access(u); T[u].self = v; pull(u);
 }
 void _Update(int u, Linear v) {
  access(u); T[u]._self = v;
  pull(u);
 }
 int FindRoot(int u) {
  access(u);
  while (T[u].ch[0]) { u = T[u].ch[0]; push(u);}
  access(u);
  return u;
 }
///k-th node (0-indexed) on the path from u to v
 int KthOnPath(int u, int v, int k) {
  if (u == v) return k == 0 ? u : -1;
  int _u = FindRoot(u);
  reroot(u); access(v);
  int ans = KthNext(u, k); reroot(_u);
  return ans;
 }
};
```

36 LIS

```
int lis(vector<int> const& a) {
 int n = a.size();
 const int INF = 1e9;
 vector<int> d(n+1, INF);
 d[0] = -INF;

 for (int i = 0; i < n; i++) {
  int j = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
  if (d[j-1] < a[i] && a[i] < d[j])
   d[j] = a[i];
 }

 int ans = 0;
 for (int i = 0; i <= n; i++) {
  if (d[i] < INF)
   ans = i;
 }
 return ans;
}
```

37 MO

```
struct query{int l,r,idx;};
int block;
bool comp1(query p,query q){
 if (p.l / block != q.l / block) {
  if(p.l==q.l) return p.r<q.r;
  return p.l < q.l;
 }
 return (p.l / block & 1) ? (p.r < q.r) : (p.r > q.r);
}
void mos_algorithm(int n, vector<query>&queries){
 vector<int> answers(queries.size());
 block = (int)sqrt(n);
 sort(queries.begin(), queries.end(),comp1);
 int cur_l = 0;
 int cur_r = -1;
 for (query q : queries) {
  while (cur_l > q.l) {cur_l--; add(cur_l);}
  while (cur_r < q.r) {cur_r++;add(cur_r);}
  while (cur_l < q.l) {Remove(cur_l);cur_l++;}
  while (cur_r > q.r) {Remove(cur_r);cur_r--;}
  answers[q.idx] = get_answer();
 }
 for(int i:answers) {cout<<i<<"\n";}
}
```

38 Manacher

```
vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
 while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
  k++;
 }
 d1[i] = k--;
 if (i + k > r){
  l = i - k;
  r = i + k;
 }
}
vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
 while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i +
      k]) {
  k++;
 }
 d2[i] = k--;
 if (i + k > r) {
  l = i - k - 1;
  r = i + k ;
 }
}
```

39 MatrixDeterminant

```
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];
}
```

40 MatrixExpo

```
/* try to avoid vector. Possibly use STL array or pointers */
void multiply(vector<vector<int> >&a, vector<vector<int> >&b){
 int n = a.size(), m= a[0].size(), l = b[0].size();
 vector<vector<int> >ret(n,vector<int>(l));
 for(int i=0; i<n; i++) {
  for(int k=0; k<m; k++) {
   for(int j=0; j<l; j++) {
    ret[i][j] = add(ret[i][j], gun(a[i][k],b[k][j],mod),mod);
   }
  }
 }
 swap(ret,a);
}

void bigmod(vector<vector<int> >&a, int p){
 int n = a.size();
 assert(a.size()==a[0].size());
 vector<vector<int> >res(n,vector<int> (n));
 for(int i=0; i<n; i++) {
  for(int j=0; j<n; j++) {
   res[i][j] = 0;
   if(i==j)
    res[i][j]=1;
  }
 }
 while(p) {
  if(p&1) {
   multiply(res,a);
  }
  multiply(a,a);
  p>>=1;
 }
 swap(a, res);
}
```

41 MillerRabin

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

u64 binpower(u64 base, u64 e, u64 mod) {
 u64 result = 1;
 base %= mod;
 while (e) {
  if (e & 1)
   result = (u128)result * base % mod;
  base = (u128)base * base % mod;
  e >>= 1;
 }
 return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
 u64 x = binpower(a, d, n);
 if (x == 1 || x == n - 1)
  return false;
 for (int r = 1; r < s; r++) {
  x = (u128)x * x % n;
  if (x == n - 1)
   return false;
 }
 return true;
};
bool MillerRabin(u64 n) { // returns true if n is prime, else
     returns false.
 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;
}
```

42 MinCostFlow1

```
#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define TEST(s) if (!(s)) { cout << __LINE__ << " " << #s <<
     endl; exit(-1); }

const long long INF = 1e9;
struct graph {
  typedef int flow_type;
  typedef int cost_type;
  struct edge {
 int src, dst;
 flow_type capacity, flow;
 cost_type cost;
 size_t rev;
  };
  void add_edge(int src, int dst, flow_type cap, cost_type
       cost) {
 adj[src].push_back({src, dst, cap, 0, cost, adj[dst].size()});
 adj[dst].push_back({dst, src, 0, 0, -cost,
      adj[src].size()-1});
  }
  int n;
  vector<vector<edge>> adj;
  graph(int n) : n(n), adj(n) { }

  pair<flow_type, cost_type> min_cost_max_flow(int s, int t) {
 flow_type flow = 0;
 cost_type cost = 0;

 for (int u = 0; u < n; ++u)
   for (auto &e: adj[u]) e.flow = 0;

 vector<cost_type> p(n, 0);

 auto rcost = [&](edge e) { return e.cost + p[e.src] -
      p[e.dst]; };
 for (int iter = 0; ; ++iter) {
   vector<int> prev(n, -1); prev[s] = 0;
   vector<cost_type> dist(n, INF); dist[s] = 0;
   if (iter == 0) {
  vector<int> count(n); count[s] = 1;
  queue<int> que;
  for (que.push(s); !que.empty(); ) {
    int u = que.front(); que.pop();
    count[u] = -count[u];
    for (auto &e: adj[u]) {
   if (e.capacity > e.flow && dist[e.dst] > dist[e.src] +
        rcost(e)) {
     dist[e.dst] = dist[e.src] + rcost(e);
     prev[e.dst] = e.rev;
     if (count[e.dst] <= 0) {
    count[e.dst] = -count[e.dst] + 1;
    que.push(e.dst);
     }
    }
   }
  } else {
  typedef pair<cost_type, int> node;
  priority_queue<node, vector<node>, greater<node>> que;
  que.push({0, s});
  while (!que.empty()) {
    node a = que.top(); que.pop();
    if (a.snd == t) break;
    if (dist[a.snd] > a.fst) continue;
    for (auto e: adj[a.snd]) {
   if (e.capacity > e.flow && dist[e.dst] > a.fst + rcost(e)) {
     dist[e.dst] = dist[e.src] + rcost(e);
     prev[e.dst] = e.rev;
     que.push({dist[e.dst], e.dst});
    }
   }
  }
   }
   if (prev[t] == -1) break;

   for (int u = 0; u < n; ++u)
  if (dist[u] < dist[t]) p[u] += dist[u] - dist[t];

   function<flow_type(int,flow_type)> augment = [&](int u,
        flow_type cur) {
  if (u == s) return cur;
  edge &r = adj[u][prev[u]], &e = adj[r.dst][r.rev];
  flow_type f = augment(e.src, min(e.capacity - e.flow, cur));
  e.flow += f; r.flow -= f;
  return f;
   };
   flow_type f = augment(t, INF);
   flow += f;
   cost += f * (p[t] - p[s]);
 }
 return {flow, cost};
  }
};
```

43 MinCostFlow2

```
struct Edge{
 int u, v;
 long long cap, cost;

 Edge(int _u, int _v, long long _cap, long long _cost) {
  u = _u;
  v = _v;
  cap = _cap;
  cost = _cost;
 }
};

struct MinCostFlow{
 int n, s, t;
 long long flow, cost;
 vector<vector<int> > graph;
 vector<Edge> e;
 /* if cost is double, dist should be double*/
 vector<long long> dist;
 vector<int> parent;

 MinCostFlow(int _n) {
  /* 0-based indexing*/
  n = _n;
  graph.assign(n, vector<int> ());
 }

 void addEdge(int u, int v, long long cap, long long cost,
      bool directed = true) {
  graph[u].push_back(e.size());
  e.push_back(Edge(u, v, cap, cost));

  graph[v].push_back(e.size());
  e.push_back(Edge(v, u, 0, -cost));

  if(!directed)
```

```
  addEdge(v, u, cap, cost, true);
 }

 pair<long long, long long> getMinCostFlow(int _s, int _t) {
  s = _s;
  t = _t;
  flow = 0, cost = 0;

  while(SPFA()) {
   flow += sendFlow(t, 1LL<<62);
  }

  return make_pair(flow, cost);
 }

 bool SPFA() {
  parent.assign(n, -1);
  dist.assign(n, 1LL<<62);
  dist[s] = 0;
  vector<int> queuetime(n, 0);
  queuetime[s] = 1;
  vector<bool> inqueue(n, 0);
  inqueue[s] = true;
  queue<int> q;
  q.push(s);
  bool negativecycle = false;

  while(!q.empty() && !negativecycle) {
   int u = q.front();
   q.pop();
   inqueue[u] = false;

   for(int i = 0; i < graph[u].size(); i++) {
    int eIdx = graph[u][i];
    int v = e[eIdx].v;
    ll w = e[eIdx].cost, cap = e[eIdx].cap;

    if(dist[u] + w < dist[v] && cap > 0) {
     dist[v] = dist[u] + w;
     parent[v] = eIdx;

     if(!inqueue[v]) {
      q.push(v);
      queuetime[v]++;
      inqueue[v] = true;

      if(queuetime[v] == n+2) {
       negativecycle = true;
       break;
      }
     }
    }
   }
  }

  return dist[t] != (1LL<<62);
 }

 long long sendFlow(int v, long long curFlow) {
  if(parent[v] == -1)
   return curFlow;
  int eIdx = parent[v];
  int u = e[eIdx].u;
  ll w = e[eIdx].cost;

  long long f = sendFlow(u, min(curFlow, e[eIdx].cap));

  cost += f*w;
  e[eIdx].cap -= f;
  e[eIdx^1].cap += f;

  return f;
 }
};
```

44 MinimumStack

```
void small_left(vector<int>& v, vector<int>& res){
 stack<pair<int, int> > stk;
 stk.push(make_pair(INT_MIN, v.size()));//initial value
 for (int i = v.size()-1; i >= 0; i--) {
  while (stk.top().first > v[i]) {
   res[stk.top().second] = i;
   stk.pop();
  }
  stk.push(make_pair(v[i], i));
 }
 while (stk.top().second < v.size()) {
  res[stk.top().second] = -1;
  stk.pop();
 }
}
```

45 Minkowski

```
void reorder_polygon(vector<pt> & P){
 size_t pos = 0;
 for(size_t i = 1; i < P.size(); i++){
  if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x <
       P[pos].x))
   pos = i;
 }
 rotate(P.begin(), P.begin() + pos, P.end());
}
vector<pt> minkowski(vector<pt> P, vector<pt> Q){
 reorder_polygon(P);reorder_polygon(Q);
 P.push_back(P[0]);P.push_back(P[1]);
 Q.push_back(Q[0]);Q.push_back(Q[1]);
 vector<pt> result;
 size_t i = 0, j = 0;
 while(i < P.size() - 2 || j < Q.size() - 2){
  result.push_back(P[i] + Q[j]);
  auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
  if(cross >= 0) ++i;
  if(cross <= 0) ++j;
 }
 return result;
}
```

46 Miscellaneous

```
for(int i=a._Find_first(); i< a.size(); i = a._Find_next(i))
mt19937 rng(chrono::steady_clock::now().
      time_since_epoch().count());
int getrand(int a, int b){
 int x = uniform_int_distribution<int>(a, b)(rng);
 return x;
}

l1.splice(l1.end(), l2);

merge(a.begin(), a.end(),
    b.begin(), b.end(),
    back_inserter(c));
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,
avx,avx2,fma")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("O3")

struct custom_hash {
 static uint64_t splitmix64(uint64_t x) {
  x += 0x9e3779b97f4a7c15;
  x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
  x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
  return x ^ (x >> 31);
 }

 size_t operator()(uint64_t x) const {
  static const uint64_t FIXED_RANDOM =
       chrono::steady_clock::now().
  time_since_epoch().count();
  return splitmix64(x + FIXED_RANDOM);
 }
};
unordered_map<long long, int, custom_hash> safe_map;

struct hash_pair {
 template <class T1, class T2>
 size_t operator()(const pair<T1, T2>& p) const {
  auto hash1 = hash<T1>{}(p.first);
  auto hash2 = hash<T2>{}(p.second);
  return hash1 ^ hash2;
 }
};
unordered_map<pair<int, int>, int, hash_pair> mp;
//mod inverse for all m
inv[1] = 1;
for(int i = 2; i < m; ++i)
 inv[i] = m - (m/i) * inv[m%i] % m;
//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>indexed_set;
//submask supermask
for (int s=m; s; s=(s-1)&m)
for (int mask=need; mask < (1<<n) ; mask = (mask+1)|need)
string str="abc def gh",buf;stringstream ss(str);
while(ss >> buf) cout << buf << endl;
```

47 Mo with update

```
const int N = 1e5 +5;
const int P = 2000; //block size = (2*n^2)^(1/3)
struct query{int t, l, r, k, i;};
vector<query> q;vector<array<int, 3>> upd;
vector<int> ans;vector<int>a;
void mos_algorithm(){
 sort(q.begin(), q.end(), [](const query &a, const query &b) {
  if (a.t / P != b.t / P) return a.t < b.t;
  if (a.l / P != b.l / P) return a.l < b.l;
  if ((a.l / P) & 1) return a.r < b.r;
  return a.r > b.r;
 });

 for (int i = upd.size() - 1; i >= 0; --i)
  a[upd[i][0]] = upd[i][1];
 int L = 0, R = -1, T = 0;
 auto apply = [&](int i, int fl) {
  int p = upd[i][0]; int x = upd[i][fl + 1];
  if (L <= p && p <= R) {rem(a[p]);add(x);}
  a[p] = x;
 };
 ans.clear();ans.resize(q.size());
 for (auto qr : q) {
  int t = qr.t, l = qr.l, r = qr.r, k = qr.k;
  while (T < t) apply(T++, 1);
  while (T > t) apply(--T, 0);
  while (R < r) add(a[++R]);
  while (L > l) add(a[--L]);
  while (R > r) rem(a[R--]);
  while (L < l) rem(a[L++]);
  ans[qr.i] = get_answer();
 }
}
void TEST_CASES(int cas){
 int n, m; cin>>n>>m;
 a.resize(n);
 for(int i=0;i<n;i++) { cin>>a[i]; }
 for(int i=0;i<m;i++) {
  int tp;scanf("%d", &tp);
  if (tp == 1) {
   int l, r, k;cin>>l>>r>>k;
   q.push_back({upd.size(), l - 1, r - 1, k, q.size()});
  }
  else {
   int p, x;cin>>p>>x;
   --p;upd.push_back({p, a[p], x});
   a[p] = x;
  }
 }
 mos_algorithm();
}
```

48 Monotone Queue

```
vector<int>dp(n,0);
vector<int>newdp(n);
for(int i=1; i<=m; i++){
int koto = d*(a[i].t - a[i-1].t);
deque<pair<int,int> >dq;
```

```
int l = 0 , r= -1;
for(int j=0; j<n; j++){
 int eil = max(0ll, j-koto) , eir = min(n-1,j+koto);
 while(!dq.empty() && dq.front().first < eil) {
  dq.pop_front();
 }
 while(r!= eir) {
  r++;
  int val = dp[r];
  int idx = r;
  while(!dq.empty() && dq.back().second <= val) {
   dq.pop_back();
  }
  dq.push_back({idx,val});
 }
 newdp[j] = a[i].b - abs(a[i].a - j-1) + dq.front().second ;
}
swap(dp,newdp);
}
cout<<*max_element(dp.begin(), dp.end())<<"\n";
```

49 NTT

```
namespace NTT {
 vector<int> perm, wp[2];
 const int mod = 998244353, G = 3; ///G is the primitive root
      of M
 int root, inv, N, invN;
 int power(int a, int p) {
  int ans = 1;
  while (p) {
   if (p & 1) ans = (1LL*ans*a)%mod;
   a = (1LL*a*a)%mod;
   p >>= 1;
  }
  return ans;
 }

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

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

  root = power(G, (mod-1)/N);
  inv = power(root, mod-2);
  wp[0][0]=wp[1][0]=1;
  for (int i=1; i<N; i++) {
   wp[0][i] = (wp[0][i-1]*1LL*root)%mod;
   wp[1][i] = (wp[1][i-1]*1LL*inv)%mod;
  }
 }
 void fft(vector<int> &v, bool invert = false) {
  if (v.size() != perm.size()) precalculate(v.size());
  for (int i=0; i<N; i++)
   if (i < perm[i])
    swap(v[i], v[perm[i]]);

  for (int len = 2; len <= N; len *= 2) {
   for (int i=0, d = N/len; i<N; i+=len) {
    for (int j=0, idx=0; j<len/2; j++, idx += d) {
     int x = v[i+j];
     int y = (wp[invert][idx]*1LL*v[i+j+len/2])%mod;
     v[i+j] = (x+y>=mod ? x+y-mod : x+y);
     v[i+j+len/2] = (x-y>=0 ? x-y : x-y+mod);
    }
   }
  }
  if (invert) {
   for (int &x : v) x = (x*1LL*invN)%mod;
  }
 }
 vector<int> multiply(vector<int> a, vector<int> b) {
  int n = 1;
  while (n < a.size()+ b.size()) n<<=1;
  a.resize(n);
  b.resize(n);

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

50 Nearest Point Pair

```
#define x first
#define y second
long long dist2(pair<int, int> a, pair<int, int> b) {
 return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y) *
      (a.y - b.y);
}
pair<int, int> closest_pair(vector<pair<int, int>> a) {
 int n = a.size();
 assert(n >= 2);
 vector<pair<pair<int, int>, int>> p(n);
 for (int i = 0; i < n; i++) p[i] = {a[i], i};
 sort(p.begin(), p.end());
 int l = 0, r = 2;
 long long ans = dist2(p[0].x, p[1].x);
 pair<int, int> ret = {0, 1};
 while (r < n) {
  while (l < r && 1LL * (p[r].x.x - p[l].x.x) * (p[r].x.x -
       p[l].x.x) >= ans) l++;
  for (int i = l; i < r; i++) {
   long long nw = dist2(p[i].x, p[r].x);
   if (nw < ans) {
    ans = nw;
    ret = {p[i].y, p[r].y};
   }
  }
  r++;
 }
 return ret;
}
```

51 PalindromicTree

```
/*-> diff(v) = len(v) - len(link(v))
-> series link will lead from the vertex v to the vertex u
     corresponding
  to the maximum suffix palindrome of v which satisfies diff(v)
       != diff(u)
-> path within series links to the root contains only O(log n)
     vertices
-> cnt contains the number of palindromic suffixes of the node*/
struct PalindromicTree {
 struct node {
  int nxt[26], len, st, en, link, diff, slink, cnt, oc;
 };
 string s;vector<node> t;
 int sz, last;
 PalindromicTree() {}
 PalindromicTree(string _s) {
  s = _s;
  int n = s.size();
  t.clear();
  t.resize(n + 9);sz = 2, last = 2;
  t[1].len = -1, t[1].link = 1;
  t[2].len = 0, t[2].link = 1;
  t[1].diff = t[2].diff = 0;
  t[1].slink = 1;t[2].slink = 2;
 }
 int extend(int pos) { // returns 1 if it creates a new
      palindrome
  int cur = last, curlen = 0;
  int ch = s[pos] - 'a';
  while (1) {curlen = t[cur].len;
   if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
        break;
   cur = t[cur].link;
  }
  if (t[cur].nxt[ch]) {last = t[cur].nxt[ch];
   t[last].oc++;
   return 0;
  }
  sz++;last = sz;
  t[sz].oc = 1;t[sz].len = t[cur].len + 2;
  t[cur].nxt[ch] = sz;t[sz].en = pos;
  t[sz].st = pos - t[sz].len + 1;
  if (t[sz].len == 1) {
   t[sz].link = 2;t[sz].cnt = 1;
   t[sz].diff = 1;t[sz].slink = 2;
   return 1;
  }
  while (1) {
   cur = t[cur].link;curlen = t[cur].len;
   if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
        {
    t[sz].link = t[cur].nxt[ch];
    break;
   }
  }
  t[sz].cnt = 1 + t[t[sz].link].cnt;
  t[sz].diff = t[sz].len - t[t[sz].link].len;
  if (t[sz].diff == t[t[sz].link].diff) t[sz].slink =
       t[t[sz].link].slink;
  else t[sz].slink = t[sz].link;
  return 1;
 }
 void calc_occurrences() {
  for (int i = sz; i >= 3; i--) t[t[i].link].oc += t[i].oc;
 }
 vector<array<int, 2>> minimum_partition() { //(even, odd), 1
      indexed
  int n = s.size();
  vector<array<int, 2>> ans(n + 1, {0, 0}), series_ans(n + 5,
       {0, 0});
  ans[0][1] = series_ans[2][1] = 1e9;
  for (int i = 1; i <= n; i++) {
   extend(i - 1);
   for (int k = 0; k < 2; k++) {
    ans[i][k] = 1e9;
    for (int v = last; t[v].len > 0; v = t[v].slink) {
     series_ans[v][!k] = ans[i - (t[t[v].slink].len +
          t[v].diff)][!k];
     if (t[v].diff == t[t[v].link].diff) series_ans[v][!k] =
          min(series_ans[v][!k], series_ans[t[v].link][!k]);
     ans[i][k] = min(ans[i][k], series_ans[v][!k] + 1);
    }
   }
  }
  return ans;
 }
} t;
int32_t main() {
 string s;cin >> s;
 PalindromicTree t(s);
 for (int i = 0; i < s.size(); i++) t.extend(i);
 t.calc_occurrences();
 long long ans = 0;
 for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
 cout << ans << '\n';
 //auto ans = t.minimum_partition();
 // for (int i = 1; i <= s.size(); i++) {
 //   cout << (ans[i][1] == 1e9 ? -1 : ans[i][1]) << ' ';
 //   cout << (ans[i][0] == 1e9 ? -2 : ans[i][0]) << '\n';
 // }
}
```

52 Partition

```
int p[MAX];
inline int Add(int a,int b){return (a+b)%mod;}

int PartitionFunction(){
 p[0] = 1;
 for(int i = 1; i < MAX; i++){
  int j = 1, r = 1;
  while(i - (3*j*j - j) / 2 >= 0){
   p[i] = Add(p[i], p[i - (3*j*j - j) / 2] * r);

   if(i - (3*j*j + j) / 2 >= 0)
```

```
    p[i] = Add(p[i], p[i - (3*j*j + j) / 2] * r);
   j += 1;
   r *= -1;
  }
  if(p[i]<0) p[i]+=mod;
 }
}
```

53 Phi

```
int phi[N+1];
void phi_1_to_n(int n) {
 phi[0] = 0;phi[1] = 1;
 for (int i = 2; i <= n; i++)phi[i] = i;
 for (int i = 2; i <= n; i++) {
  if (phi[i] == i) { for (int j = i; j <= n; j += i)
    phi[j] -= phi[j] / i;
  }
 }
}
```

54 PointOrderingByAngle

```
typedef pair<int,int> pii;
struct point{
 int x, y;
 bool operator <(point &p) const{
  if(x==p.x){
   return y > p.y;
  }
  return x<p.x;
 }
};
struct line{
 point p1, p2;
 line(){}
 line(point &p, point &q){
  p1 = p;
  p2 = q;
 }
 bool operator <(line &p){
  ll a = (p1.x- p2.x);
  ll b = (p1.y-p2.y);
  ll c = (p.p1.x- p.p2.x);
  ll d = (p.p1.y-p.p2.y);
  return a*d < b*c;
 }
};
void TEST_CASES(int cas){
 int n;
 scanf("%d",&n);
 map<pii, int>mp;
 vector<point>v(n);
 for(int i=0; i<n; i++){
  v[i].read();
 }
 sort(v.begin(),v.end());
 for(int i=0; i<n; i++){
  mp[ {v[i].x, v[i].y}] = i;
 }
 vector<line>lines;
 for(int i=0; i<n; i++){
  for(int j=i+1; j<n; j++){
   lines.emplace_back(v[i],v[j]);
  }
 }
 sort(lines.begin(),lines.end());
 for(line &l: lines){
  point p1= l.p1;
  point p2 = l.p2;
  int idx1 = mp[ {p1.x,p1.y}];
  int idx2 = mp[ {p2.x, p2.y}];
  //Do your work
  //Swap indexes
  v[idx1] = l.p2;
  v[idx2] = l.p1;
  mp[ {v[idx1].x, v[idx1].y}] = idx1;
  mp[ {v[idx2].x, v[idx2].y}] = idx2;
 }
}
```

55 PollardRho

```
typedef long long LL;
typedef unsigned long long ULL;
namespace Rho {
 ULL mult(ULL a, ULL b, ULL mod) {
  LL ret = a * b - mod * (ULL)(1.0L / mod * a * b);
  return ret + mod * (ret < 0) - mod * (ret >= (LL) mod);
 }
 ULL power(ULL x, ULL p, ULL mod){
  ULL s=1, m=x;
  while(p) {
   if(p&1) s = mult(s, m, mod);
   p>>=1;
   m = mult(m, m, mod);
  }
  return s;
 }
 vector<LL> bases = {2, 325, 9375, 28178, 450775, 9780504,
      1795265022};
 bool isprime(LL n) {
  if (n<2) return 0;
  if (n%2==0)  return n==2;

  ULL s = __builtin_ctzll(n-1), d = n>>s;
  for (ULL x: bases) {
   ULL p = power(x%n, d, n), t = s;
   while (p!=1 && p!=n-1 && x%n && t--) p = mult(p, p, n);
   if (p!=n-1 && t != s)    return 0;
  }
  return 1;
 }
 mt19937_64 rng(chrono::system_clock::now().
 time_since_epoch().count());
 ULL FindFactor(ULL n) {
  if (n == 1 || isprime(n)) return n;
  ULL c = 1, x = 0, y = 0, t = 0, prod = 2, x0 = 1, q;
  auto f = [&](ULL X) { return mult(X, X, n) + c;};

  while (t++ % 128 or gcd(prod, n) == 1) {
   if (x == y) c = rng()%(n-1)+1, x = x0, y = f(x);
   if ((q = mult(prod, max(x, y) - min(x, y), n))) prod = q;
   x = f(x), y = f(f(y));
  }
  return gcd(prod, n);
 }
 vector<ULL> factorize(ULL x) {
  if (x == 1) return {};
  ULL a = FindFactor(x), b = x/a;
  if (a == x) return {a};
  vector<ULL> L = factorize(a), R = factorize(b);
  L.insert(L.end(), R.begin(), R.end());
  return L;
 }
}
```

56 Popcount Trick

```
int cnt[1<<16];
void preprocess(){
    for(int mask=0;mask<(1<<16);mask++) {
        for(int i=0;i<16;i++) {
            if(mask&(1<<i)) cnt[mask]++;
        }
    }
}
int query(ll x){
    int ans=0,G=(1<<16)-1;
    ans+=cnt[x&G];x>>=16;
    ans+=cnt[x&G];x>>=16;
    ans+=cnt[x&G];x>>=16;
    ans+=cnt[x&G];
    return ans;
}
```

57 PrefixFunction

```
vector<int> prefix_function(string s) {
 int n = (int)s.length();
 vector<int> pi(n);
 for (int i = 1; i < n; i++) {
  int j = pi[i-1];
  while (j > 0 && s[i] != s[j])
   j = pi[j-1];
  if (s[i] == s[j])
   j++;
  pi[i] = j;
 }
 return pi;
}
```

58 PrimitiveRoot

```
// Finds the primitive root modulo p
int generator(int p) {
 vector<int> fact;
 int phi = p-1, n = phi;
 for (int i = 2; i * i <= n; ++i) {
  if (n % i == 0) {
   fact.push_back(i);
   while (n % i == 0)
    n /= i;
  }
 }
 if (n > 1) fact.push_back(n);
 for (int res = 2; res <= p; ++res) {
  bool ok = true;
  for (int factor : fact) {
   if (powmod(res, phi / factor, p) == 1) {
    ok = false; break;
   }
  }
  if (ok) return res;
 }
 return -1;
}
// This program finds all numbers x such that x^k = a (mod n)
int main() {
 int n, k, a;
 scanf("%d %d %d", &n, &k, &a);
 if (a == 0) {
  puts("1\n0");
  return 0;
 }
 int g = generator(n);
 int sq = (int) sqrt (n + .0) + 1;
 vector<pair<int, int>> dec(sq);
 for (int i = 1; i <= sq; ++i)
  dec[i-1] = {powmod(g, i * sq * k % (n - 1), n), i};
 sort(dec.begin(), dec.end());
 int any_ans = -1;
 for (int i = 0; i < sq; ++i) {
  int my = powmod(g, i * k % (n - 1), n) * a % n;
  auto it = lower_bound(dec.begin(), dec.end(), make_pair(my,
       0));
  if (it != dec.end() && it->first == my) {
   any_ans = it->second * sq - i;
   break;
  }
 }
 if (any_ans == -1) {
  puts("0");return 0;
 }
 // Print all possible answers
 int delta = (n-1) / gcd(k, n-1);
 vector<int> ans;
 for (int cur = any_ans % delta; cur < n-1; cur += delta)
  ans.push_back(powmod(g, cur, n));
 sort(ans.begin(), ans.end());
}
```

59 SCC

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;
```

```
void dfs1(int v) {
 used[v] = true;
 for (auto u : adj[v])
  if (!used[u]) dfs1(u);
 order.push_back(v);
}
void dfs2(int v) {
 used[v] = true; component.push_back(v);
 for (auto u : adj_rev[v]) if (!used[u])
   dfs2(u);
}
int main() {
 used.assign(n, false);
 for (int i = 0; i < n; i++)
  if (!used[i]) dfs1(i);
 used.assign(n, false);
 reverse(order.begin(), order.end());
 for (auto v : order) if (!used[v]) {
   dfs2 (v);
   //processing next component
   component.clear();
  }
}
```

60 SOSDP

```
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
 F[i] = A[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)];
 }
}

//supermask
for(int i = 0; i < N; ++i){
 for(int mask = (1<<N)-1; mask >=0; --mask) {
  if(!(mask & (1<<i)))
   dp[mask] += dp[mask|(1<<i)];
 }
}
```

61 SegSegIntersection

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

62 SegmentedSieve

```
vector<char> segmentedSieve(long long L, long long R) {
 // generate all primes up to sqrt(R)
 long long lim = sqrt(R);
 vector<char> mark(lim + 1, false);
 vector<long long> primes;
 for (long long i = 2; i <= lim; ++i) {
  if (!mark[i]) {
   primes.emplace_back(i);
   for (long long j = i * i; j <= lim; j += i)
    mark[j] = true;
  }
 }

 vector<char> isPrime(R - L + 1, true);
 for (long long i : primes)
  for (long long j = max(i * i, (L + i - 1) / i * i); j <= R;
       j += i)
   isPrime[j - L] = false;
 if (L == 1)
  isPrime[0] = false;
 return isPrime;
}
```

63 Segtree 2D

```
void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
 if (ly == ry) {
  if (lx == rx)
   t[vx][vy] = a[lx][ly];
  else
   t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
 } else {
  int my = (ly + ry) / 2;
  build_y(vx, lx, rx, vy*2, ly, my);
  build_y(vx, lx, rx, vy*2+1, my+1, ry);
  t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
 }
}

void build_x(int vx, int lx, int rx) {
 if (lx != rx) {
  int mx = (lx + rx) / 2;
  build_x(vx*2, lx, mx);
  build_x(vx*2+1, mx+1, rx);
 }
 build_y(vx, lx, rx, 1, 0, m-1);
}
int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
 if (ly > ry)
  return 0;
 if (ly == tly && try_ == ry)
  return t[vx][vy];
 int tmy = (tly + try_) / 2;
 return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
   + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
}

int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly,
     int ry) {
 if (lx > rx)
  return 0;
 if (lx == tlx && trx == rx)
  return sum_y(vx, 1, 0, m-1, ly, ry);
 int tmx = (tlx + trx) / 2;
 return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
   + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
}
void update_y(int vx, int lx, int rx, int vy, int ly, int ry,
     int x, int y, int new_val) {
 if (ly == ry) {
  if (lx == rx)
   t[vx][vy] = new_val;
  else
   t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
 } else {
  int my = (ly + ry) / 2;
  if (y <= my)
   update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
  else
   update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
  t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
 }
}

void update_x(int vx, int lx, int rx, int x, int y, int
     new_val) {
 if (lx != rx) {
  int mx = (lx + rx) / 2;
  if (x <= mx)
   update_x(vx*2, lx, mx, x, y, new_val);
  else
   update_x(vx*2+1, mx+1, rx, x, y, new_val);
 }
 update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
```

64 Segtree Lazy

```
void push(int v) {
 t[v*2] += lazy[v];
 lazy[v*2] += lazy[v];
 t[v*2+1] += lazy[v];
 lazy[v*2+1] += lazy[v];
 lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int r, int addend) {
 if (l > r)
  return;
 if (l == tl && tr == r) {
  t[v] += addend;
  lazy[v] += addend;
 } else {
  push(v);
  int tm = (tl + tr) / 2;
  update(v*2, tl, tm, l, min(r, tm), addend);
  update(v*2+1, tm+1, tr, max(l, tm+1), r, addend);
  t[v] = max(t[v*2], t[v*2+1]);
 }
}

int query(int v, int tl, int tr, int l, int r) {
 if (l > r)
  return -INF;
 if (l <= tl && tr <= r)
  return t[v];
 push(v);
 int tm = (tl + tr) / 2;
 return max(query(v*2, tl, tm, l, min(r, tm)),
      query(v*2+1, tm+1, tr, max(l, tm+1), r));
}
```

65 Segtree Persistent

```
const ll INF = 4e18;
//Point update and range min
struct Node {
 int l=-1,r=-1; pii val;
 Node(pii v, int l=-1, int r=-1) {
  this->l = l;
  this->r = r;
  val = v;
 }
};
vector<Node>nodes;
inline void Merge(Node &a, Node &b, Node &c) {
 a.val = min(b.val,c.val);
}
int build(int tl, int tr) {
 if (tl == tr) {
  nodes.emplace_back(make_pair(INF, -1));
  return (int)nodes.size()-1;
 }
 int tm = (tl + tr) / 2;
 int Left = build(tl, tm);
 int Right = build(tm+1, tr);
 nodes.emplace_back(make_pair(INF, -1),Left,Right);
 Merge(nodes.back(), nodes[Left], nodes[Right]);
```

```
 return (int)nodes.size()-1;
}

int update(int v, int tl, int tr, int pos, pii val) {
 if (tl == tr) {
  nodes.emplace_back(val);
  return (int)nodes.size()-1;
 }
 int tm = (tl + tr) / 2; int Left,Right;
 if (pos <= tm) {
  Left = update(nodes[v].l, tl, tm, pos,val);
  Right = nodes[v].r;
 }
 else {
  Left = nodes[v].l;
  Right = update(nodes[v].r, tm+1, tr, pos,val);
 }
 nodes.emplace_back(make_pair(INF, -1),Left,Right);
 Merge(nodes.back(), nodes[Left], nodes[Right]);
 return (int)nodes.size()-1;
}

pii query(int v, int tl, int tr, int a, int b) {
 if (a>tr|| tl>b || tl > tr) return {INF, -1};
 if(tl >=a && tr <=b) return nodes[v].val;
 int mid= (tl+tr)/2;
 return min(query(nodes[v].l,tl, mid, a,b), query(nodes[v].r,
      mid+1, tr, a, b));
}
```

66 Segtree beats

```
struct info{
 int maxi = 0,smaxi = -1e9, cnt = 0, lazy = 0;
 bool has=0;ll sum = 0;
};
struct segtree{
 int n;
 vector<info>t;
 segtree(int n,vector<int>&a)
 {
  this->n = n;t.resize(n*4);
  build(1,0,n-1,a);
 }
 void Merge(info &node, info &l, info &r){
  node.maxi = max(l.maxi, r.maxi) ;
  node.cnt = (node.maxi==l.maxi ? l.cnt : 0) +
       (node.maxi==r.maxi ? r.cnt : 0);
  node.sum = l.sum + r.sum;
  if(l.maxi != r.maxi)
   node.smaxi =max({ min(l.maxi, r.maxi) ,l.smaxi, r.smaxi});
  else
   node.smaxi = max(l.smaxi, r.smaxi);
 }
 void build(int node, int l,int r, vector<int>&a) {
  if(l==r){
   t[node].maxi = a[l];t[node].cnt = 1;t[node].sum = a[l];
   return;
  }
  int mid= (l+r)/2;
  build(node*2, l, mid,a);
  build(node *2 +1, mid+1, r,a);
  Merge(t[node], t[node*2], t[node *2+1]);
 }
 void dop(int node, int add){
  if(t[node].maxi <= add)
   return;
  t[node].sum -= t[node].maxi *1ll* t[node].cnt;
  t[node].maxi = add;
  t[node].sum += t[node].maxi *1ll* t[node].cnt;
  t[node].lazy= add;
  t[node].has = 1;
 }
 void push_down(int node){
  if(t[node].has){
   dop(node*2, t[node].lazy);
   dop(node*2+1, t[node].lazy);
   t[node].lazy =0;
   t[node].has = 0;
  }
 }
 void update(int node, int l,int r, int i, int j, int add){
  if(l>j || r<i || t[node].maxi <= add){
   return;
  }
  if(l>=i && r<=j && t[node].smaxi < add){
   int x = t[node].maxi - add;
   t[node].sum -= t[node].maxi *1ll* t[node].cnt;
   t[node].maxi = add;
   t[node].sum += t[node].maxi *1ll* t[node].cnt;
   t[node].lazy= add;
   t[node].has = 1;
   return;
  }
  int mid = (l+r)/2;
  push_down(node);
  update(node *2, l,mid, i, j, add);
  update(node *2 +1, mid+1, r, i, j, add);
  Merge(t[node], t[node *2], t[node*2+1]);
 }
 void update(int l,int r, int add){
  update(1,0,n-1,l,r,add);
 }
 pair<ll,int> query(int node, int l, int r, int i, int j){
  if(l>j || r<i){
   return make_pair(0,-1e9);
  }
  if(l>=i && r<=j){
   return make_pair(t[node].sum,t[node].maxi);
  }
  int mid = (l+r)/2;
  push_down(node);
  pair<ll,int> x = query(node *2, l,mid, i, j);
  pair<ll,int> y = query(node *2 +1, mid+1, r, i, j);
  return make_pair(x.first+y.first , max(x.second,y.second));
 }
 pair<ll,int> query(int l, int r){
  return query(1,0,n-1,l,r ) ;
 }
};
```

67 Simplex

```
/*
 * Note: Simplex algorithm on augmented matrix a of dimension
      (m+1)x(n+1)
 * returns 1 if feasible, 0 if not feasible, -1 if unbounded
 * returns solution in b[] in original var order, max(f) in ret
 * form: maximize sum_j(a_mj*x_j)-a_mn s.t.
      sum_j(a_ij*x_j)<=a_in
 * in standard form.
 * To convert into standard form:
 * 1. if exists equality constraint, then replace by both >=
      and <=
 * 2. if variable x doesn't have nonnegativity constraint, then
      replace by
 * difference of 2 variables like x1-x2, where x1>=0, x2>=0
 * 3. for a>=b constraints, convert to -a<=-b
 * note: watch out for -0.0 in the solution, algorithm may cycle
 * EPS = 1e-7 may give wrong answer, 1e-10 is better
 */
typedef vector<ld> vd;
typedef vector<vd> vvd;
typedef vector<int> vi;
const ld EPS = 1e-10;
struct LPSolver{
 int m, n; vi B, N; vvd D;
 LPSolver(const vvd &A, const vd &b, const vd &c) :
  m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vd(n +
       2)) {
  for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++)
    D[i][j] = A[i][j];
  for (int i = 0; i < m; i++) {
   B[i] = n + i; D[i][n] = -1;
   D[i][n + 1] = b[i];
  }
  for (int j = 0; j < n; j++) {
   N[j] = j; D[m][j] = -c[j];
  }
  N[n] = -1; D[m + 1][n] = 1;
 }
 void Pivot(int r, int s) {
  ld inv = 1.0 / D[r][s];
  for (int i = 0; i < m + 2; i++) if (i != r)
    for (int j = 0; j < n + 2; j++)
     if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
  for (int j = 0; j < n + 2; j++) if (j != s)
    D[r][j] *= inv;
  for (int i = 0; i < m + 2; i++) if (i != r)
    D[i][s] *= -inv;
  D[r][s] = inv; swap(B[r], N[s]);
 }
 bool Simplex(int phase) {
  int x = phase == 1 ? m + 1 : m;
  while (true) {
   int s = -1;
   for (int j = 0; j <= n; j++) {
    if (phase == 2 && N[j] == -1) continue;
    if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] &&
         N[j] < N[s])
     s = j;
   }
   if (D[x][s] > -EPS) return true;
   int r = -1;
   for (int i = 0; i < m; i++) {
    if (D[i][s] < EPS) continue;
    if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] /
         D[r][s] ||
      (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) &&
          B[i] < B[r])
     r = i;
   }
   if (r == -1) return false;
   Pivot(r, s);
  }
 }
 ld Solve(vd &x) {
  int r = 0;
  for (int i = 1; i < m; i++)
   if (D[i][n + 1] < D[r][n + 1])
    r = i;
  if (D[r][n + 1] < -EPS) {
   Pivot(r, n);
   if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
    return -numeric_limits<ld>::infinity();
   for (int i = 0; i < m; i++)
    if (B[i] == -1) {
     int s = -1;
     for (int j = 0; j <= n; j++)
      if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] &&
           N[j] < N[s])
       s = j;
     Pivot(i, s);
    }
  }
  if (!Simplex(2))
   return numeric_limits<ld>::infinity();
  x = vd(n);
  for (int i = 0; i < m; i++)
   if (B[i] < n) x[B[i]] = D[i][n + 1];
  return D[m][n + 1];
 }
};
/* Equations are of the matrix form Ax<=b, and we want to
     maximize
the function c. We are given coeffs of A, b and c. In case of
     minimizing,
we negate the coeffs of c and maximize it. Then the negative of
     returned
'value' is the answer.
All the constraints should be in <= form. So we may need to
     negate the
coeffs.
*/
int main(){
 const int m = 4; const int n = 3;
 ld _A[m][n] = { { 6, -1, 0 }, { -1, -5, 0 }, { 1, 5, 1 }, {
      -1, -5, -1 }
```

```
 };
 ld _b[m] = { 10, -4, 5, -5 };
 ld _c[n] = { 1, -1, 0 };
 vvd A(m);
 vd b(_b, _b + m);
 vd c(_c, _c + n);
 for (int i = 0; i < m; i++)
  A[i] = vd(_A[i], _A[i] + n);
 LPSolver solver(A, b, c);
 vd x;
 ld value = solver.Solve(x);
 cerr << "VALUE: " << value << endl; /* VALUE: 1.29032*/
 cerr << "SOLUTION:"; /*SOLUTION: 1.74194 0.451613 1*/
 for (size_t i = 0; i < x.size(); i++)
  cerr << " " << x[i];
}
```

68 Simpson

```
const int N = 1000 * 1000; // number of steps (already
     multiplied by 2)

double simpson_integration(double a, double b){
 double h = (b - a) / N;
 double s = f(a) + f(b); // a = x_0 and b = x_2n
 for (int i = 1; i <= N - 1; ++i) { // Refer to final
      Simpson's formula
  double x = a + h * i;
  s += f(x) * ((i & 1) ? 4 : 2);
 }
 s *= h / 3;
 return s;
}
```

69 StableMarriage

```
//order[i][j]=indexOfMan i in j-th women'sListOfPreference
//prefer[i]=listOfWomen inOrderOf decreasingPreference
int n;
int pre[N][N],order[N][N],nxt[N];
queue<int>q;
int future_wife[N],future_husband[N];
void engage(int man , int woman){
 int m1 = future_husband[woman];
 if(m1==0) {
  future_wife[man] = woman;future_husband[woman] = man;
 }
 else{
  future_wife[man] = woman;future_husband[woman] = man;
  future_wife[m1] =0;q.push(m1);
 }
}

void TEST_CASES(int cas){
 while(!q.empty())q.pop();
 cin>>n;
 for(int i=1;i<=n;i++) {
  for(int j=1;j<=n;j++) {
   cin>>pre[i][j]; pre[i][j]-= n;
  }
  nxt[i] = 1;future_wife[i] = 0;q.push(i);
 }
 for(int i=1;i<=n;i++) {
  for(int j=1;j<=n;j++) {
   int x; cin>>x;
   order[i][x] = j;
  }
  future_husband[i] = 0;
 }
 while(!q.empty()) {
  int man = q.front(); q.pop();
  int woman = pre[man][nxt[man]++];
  if(future_husband[woman]==0) {
   engage(man , woman);
  }
  else if(order[woman][man] <
       order[woman][future_husband[woman]]) {
   engage(man , woman);
  }
  else{ q.push(man); }
 }
 for(int i=1;i<=n;i++) {
  cout<<" ("<<i<<" "<<future_wife[i]+n<<")";
 }
}
```

70 Stirling

```
NTT ntt(mod);
vector<ll>v[MAX];

//Stirling1 (n,k) = co-eff of x^k in x*(x+1)*(x+2)*....(x+n-1)
int Stirling1(int n, int r) {
 int nn = 1;
 while(nn < n) nn <<= 1;

 for(int i = 0; i < n; ++i) {v[i].push_back(i);
      v[i].push_back(1);}
 for(int i = n; i < nn; ++i) v[i].push_back(1);

 for(int j = nn; j > 1; j >>= 1) {
  int hn = j >> 1;
  for(int i = 0; i < hn; ++i) ntt.multiply(v[i], v[i + hn],
       v[i]);
 }
 return v[0][r];
}

NTT ntt(mod);
vector<int>a,b,res;

//Stirling2 (n,k) = co-eff of x^k in product of polynomials A &
     B
//where A(i) = (-1)^i / i! and B(i) = i^n / i!
int Stirling2(int n, int r) {
 a.resize(n+1); b.resize(n+1);
 for(int i = 0; i <= n; i++){
  a[i] = invfct[i];
  if(i % 2 == 1) a[i] = mod - a[i];
 }

 for(int i = 0; i <= n; i++){
  b[i] = bigMod(i, n, mod);
  b[i] = (b[i] *1ll* invfct[i]) % mod;
 }

 NTT ntt(mod);
 ntt.multiply(a,b,res);
 return res[r];
}
```

71 StressTest

```
#!/bin/sh
echo "Enter the name of first File : "
read file
echo "Enter the name of second File : "
read file2
g++ -o test_gen test_gen.cpp
g++ -o $file $file.cpp
g++ -o $file2 $file2.cpp
while true
do
   ./test_gen
   ./$file <input.txt> out1.txt
   ./$file2 <input.txt> out2.txt
   if cmp -s "out1.txt" "out2.txt"; then
 echo "Test Case OK"
   else
 echo "ERROR ENCOUNTERED"
 break
   fi
done
```

72 Suffix Automata

```
struct state{
 int len, link;
 map<char, int> next;
 ll dp=-1; //number of paths
 ll cnt=0; //endpos size
 bool is_cloned=false;
 vector<int>inv_link;
};
struct SA {
 vector<state> st;
 int sz, last;
 void sa_init() {
  st[0].len = 0;
  st[0].link = -1;
  sz=0;
  sz++;
  last = 0;
 }
 void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p = last;
  while (p != -1 && !st[p].next.count(c)) {
   st[p].next[c] = cur;
   p = st[p].link;
  }
  if (p == -1) {
   st[cur].link = 0;
  }
  else {
   int q = st[p].next[c];
   if (st[p].len + 1 == st[q].len){
    st[cur].link = q;
   }
   else {
    int clone = sz++;
    st[clone].len = st[p].len + 1;
    st[clone].next = st[q].next;
    st[clone].link = st[q].link;
    while (p != -1 && st[p].next[c] == q) {
     st[p].next[c] = clone;
     p = st[p].link;
    }
    st[q].link = st[cur].link = clone;
    st[clone].is_cloned=true;
   }
  }
  last = cur;
 }
 ll run(int idx) {
  if(st[idx].dp!=-1)
   return st[idx].dp;
  if(idx!=0)
   st[idx].dp=st[idx].cnt;
  else st[idx].dp=0;
  for(char c='a';c<='z';c++) {
   if(!st[idx].next.count(c))
    continue;
   int u=st[idx].next[c];
   st[idx].dp+=run(u);
  }
  return st[idx].dp;
 }
 void dfs_in_tree(int idx) {
  if(st[idx].is_cloned==false) {
   st[idx].cnt=1;
  }
  for(int u:st[idx].inv_link) {
   dfs_in_tree(u);
   st[idx].cnt+=st[u].cnt;
  }
 }
 void build(string &s) {
  st.resize(2*(int)s.size());
  sa_init();
  for(char c:s) {
   sa_extend(c);
  }
  for(int i=1;i<sz;i++) {
   st[st[i].link].inv_link.push_back(i);
  }
  dfs_in_tree(0);
 }
};
```

73 SuffixArray

```
const int LOG = 20;
int sa[N],Data[N],rnk[N],height[N];
int wa[N],wb[N],wws[N],wv[N];
int lg[N], rmq[N][LOG];
void prelg() {
 lg[0] = lg[1] = 0;
 for(int i = 2; i < N; i++) {
  lg[i] = lg[i/2] + 1;
 }
}
struct SuffixArray {
 int n;
 int cmp(int *r,int a,int b,int l) {
  return (r[a]==r[b]) && (r[a+l]==r[b+l]);
 }
 void DA(int *r,int *sa,int n,int m) {
  int i,j,p,*x=wa,*y=wb,*t;
  for(i=0; i<m; i++) wws[i]=0;
  for(i=0; i<n; i++) wws[x[i]=r[i]]++;
  for(i=1; i<m; i++) wws[i]+=wws[i-1];
  for(i=n-1; i>=0; i--) sa[--wws[x[i]]]=i;
  for(j=1,p=1; p<n; j*=2,m=p)
  {
   for(p=0,i=n-j; i<n; i++) y[p++]=i;
   for(i=0; i<n; i++)
    if(sa[i]>=j) y[p++]=sa[i]-j;
   for(i=0; i<n; i++) wv[i]=x[y[i]];
   for(i=0; i<m; i++) wws[i]=0;
   for(i=0; i<n; i++) wws[wv[i]]++;
   for(i=1; i<m; i++) wws[i]+=wws[i-1];
   for(i=n-1; i>=0; i--) sa[--wws[wv[i]]]=y[i];
   for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)
    x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
  }
 }
 void calheight(int *r,int *sa,int n) {
  int i,j,k=0;
  for(i=1; i<=n; i++) rnk[sa[i]]=i;
  for(i=0; i<n; height[rnk[i++]]=k)
   for(k?k--:0,j=sa[rnk[i]-1]; r[i+k]==r[j+k]; k++);
 }
 void suffix_array (string &A) {
  n = A.size();
  Data[n]=0;
  int cnt =0;
  for (int i = 0; i < n; i++){
   Data[i] = A[i]-'a'+1; //careful
   cnt = max(cnt, Data[i]);
  }
  DA(Data,sa,n+1,cnt+1);
  calheight(Data,sa,n);
  for(int i = 0; i < n; i++)
   sa[i] = sa[i+1], height[i] = height[i+1], rnk[sa[i]] = i;
  range_lcp_init();
 }
 /** LCP for range : build of rmq table **/
 void range_lcp_init() {
  for(int i = 0; i < n; i++)
   rmq[i][0] = height[i];
  for(int j = 1; j < LOG; j++) {
   for(int i = 0; i < n; i++) {
    if (i+(1<<j)-1 < n)
     rmq[i][j] = min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
    else break;
   }
  }
 }
 /** lcp between l'th to r'th suffix in suffix array **/
 int query_lcp(int l, int r) {
  assert(l <= r); assert(l>=0 && l<n && r>=0 && r<n);
  if(l == r) return n-sa[l];
  l++;
  int k = lg[r-l+1];
  return min(rmq[l][k],rmq[r-(1<<k)+1][k]);
 }
 //i and j position in original string
 int getsuff(int i, int j) {
  i= rnk[i];j = rnk[j];
  return query_lcp(min(i,j),max(i,j));
 }
} SA;
```

74 Treap

```
mt19937 rng(chrono::steady_clock::now().
time_since_epoch().count());
int getrand(int a, int b){
 int x = uniform_int_distribution<int>(a, b)(rng);
 return x;
}
struct treap{
 int prior,val,subtreeSize ;
 treap *l;treap *r;treap *parent;
 int sum,lazy;
 treap(int data) {
  val= data;prior = getrand(-2e9, 2e9);
  subtreeSize = 1;
  l=NULL;r=NULL;parent = NULL;lazy = 0;sum = data;
 }
};
typedef treap* ptreap;
int Size(ptreap t){
 if(t)return t->subtreeSize;
 return 0;
}
void update_size(ptreap t){
 if(t) t->subtreeSize = 1+ Size(t->l) + Size(t->r);
}
void push(ptreap t){
 if(!t || !t->lazy) {return;}
 t->val += t->lazy;t->sum += t->lazy * Size(t);
 if(t->l) t->l->lazy += t->lazy;
 if(t->r) t->r->lazy += t->lazy;
 t->lazy = 0;
}
void reset(ptreap t){
 if(t) t->sum = t->val;
}
void combine(ptreap &t, ptreap l, ptreap r){
 if(!l || !r) {if(l) {t= l;}else {t = r;}return;
 }
 t->sum = l->sum + r->sum;
}
void operation(ptreap t){
 if(!t)return;
 reset(t);push(t->l);push(t->r);
 combine(t,t,t->l);combine(t,t,t->r);
}
void split(ptreap t, ptreap &l, ptreap &r, int pos, int add
    =0){
 if(!t) {l = NULL;r = NULL;return;}
 push(t);
 int curr = add + Size(t->l);
 if(curr<=pos) {
  split(t->r, t->r, r, pos, curr+1);
  if(t->r != NULL) t->r->parent = t;
  if(r!=NULL) r->parent = NULL;
  l = t;
 }
 else {
  split(t->l, l, t->l,pos, add);
  if(t->l != NULL) {t->l->parent = t;
  }
  if(l!=NULL) {l->parent = NULL;}
  r =t;
 }
 update_size(t);operation(t);
}
void Merge(ptreap &t, ptreap l, ptreap r){
 push(l);push(r);
 if(!l || !r) {if(l) t= l;else t = r;}
 else if(l->prior > r->prior) {
  Merge(l->r, l->r, r);
  if(l->r != NULL) {l->r ->parent = l;}
  t= l;
 }
 else {
  Merge(r->l, l, r->l);
  if(r->l != NULL) {r->l->parent = r;}
  t = r;
 }
 update_size(t);operation(t);
}
int range_query(ptreap t, int l, int r){
 ptreap t1, t2, t3;
 split(t,t1,t2,l-1);split(t2,t2,t3,r-l);
 int ans = t2->sum;
 Merge(t,t1,t2);Merge(t,t,t3);
 return ans;
}
void range_update(ptreap t, int l,int r, int val){
 ptreap t1, t2, t3;
 split(t,t1,t2,l-1);split(t2,t2,t3,r-l);
 t2->lazy += val;Merge(t,t1,t2);Merge(t,t,t3);
}
ptreap goup(ptreap t){
 if(t==NULL || t->parent==NULL) return t;
 return goup(t->parent);
}
void output2 (ptreap t){
 if (!t) return;
 push (t);output2 (t->l);
 cout<<t->val<<" ";output2 (t->r);
}
```

75 VerticalDecomposition

```
inline bool le(dbl x, dbl y){return x < y + eps;}
inline bool ge(dbl x, dbl y){return x > y - eps;}
struct Line{
 pt p[2];
 Line(){}
 Line(pt a, pt b):p{a, b}{}
 pt vec()const{
  return p[1] - p[0];
 }
 pt& operator [](size_t i){
  return p[i];
 }
};

inline bool lexComp(const pt & l, const pt & r){
 if(fabs(l.x - r.x) > eps){return l.x < r.x;}
 else return l.y < r.y;
}

vector<pt> interSegSeg(Line l1, Line l2){
 if(eq(l1.vec().cross(l2.vec()), 0)){
  if(!eq(l1.vec().cross(l2[0] - l1[0]), 0))
   return {};
  if(!lexComp(l1[0], l1[1])) swap(l1[0], l1[1]);
  if(!lexComp(l2[0], l2[1])) swap(l2[0], l2[1]);
  pt l = lexComp(l1[0], l2[0]) ? l2[0] : l1[0];
  pt r = lexComp(l1[1], l2[1]) ? l1[1] : l2[1];
  if(l == r)
   return {l};
  else return lexComp(l, r) ? vector<pt>{l, r} : vector<pt>();
 }
 else{
  dbl s = (l2[0] - l1[0]).cross(l2.vec()) /
       l1.vec().cross(l2.vec());
  pt inter = l1[0] + l1.vec() * s;
  if(ge(s, 0) && le(s, 1) && le((l2[0] - inter).dot(l2[1] -
       inter), 0))
   return {inter};
  else
   return {};
 }
}
inline char get_segtype(Line segment, pt other_point){
 if(eq(segment[0].x, segment[1].x))
  return 0;
 if(!lexComp(segment[0], segment[1]))
  swap(segment[0], segment[1]);
 return (segment[1] - segment[0]).cross(other_point -
      segment[0]) > 0 ? 1 : -1;
```

```
}

dbl union_area(vector<tuple<pt, pt, pt> > triangles){
 vector<Line> segments(3 * triangles.size());
 vector<char> segtype(segments.size());
 for(size_t i = 0; i < triangles.size(); i++){
  pt a, b, c;
  tie(a, b, c) = triangles[i];
  segments[3 * i] = lexComp(a, b) ? Line(a, b) : Line(b, a);
  segtype[3 * i] = get_segtype(segments[3 * i], c);
  segments[3 * i + 1] = lexComp(b, c) ? Line(b, c) : Line(c,
       b);
  segtype[3 * i + 1] = get_segtype(segments[3 * i + 1], a);
  segments[3 * i + 2] = lexComp(c, a) ? Line(c, a) : Line(a,
       c);
  segtype[3 * i + 2] = get_segtype(segments[3 * i + 2], b);
 }
 vector<dbl> k(segments.size()), b(segments.size());
 for(size_t i = 0; i < segments.size(); i++){
  if(segtype[i]){
   k[i] = (segments[i][1].y - segments[i][0].y) /
        (segments[i][1].x - segments[i][0].x);
   b[i] = segments[i][0].y - k[i] * segments[i][0].x;
  }
 }
 dbl ans = 0;
 for(size_t i = 0; i < segments.size(); i++){
  if(!segtype[i])
   continue;
  dbl l = segments[i][0].x, r = segments[i][1].x;
  vector<pair<dbl, int> > evts;
  for(size_t j = 0; j < segments.size(); j++){
   if(!segtype[j] || i == j)
    continue;
   dbl l1 = segments[j][0].x, r1 = segments[j][1].x;
   if(ge(l1, r) || ge(l, r1))
    continue;
   dbl common_l = max(l, l1), common_r = min(r, r1);
   auto pts = interSegSeg(segments[i], segments[j]);
   if(pts.empty()){
    dbl yl1 = k[j] * common_l + b[j];
    dbl yl = k[i] * common_l + b[i];
    if(lt(yl1, yl) == (segtype[i] == 1)){
     int evt_type = -segtype[i] * segtype[j];
     evts.emplace_back(common_l, evt_type);
     evts.emplace_back(common_r, -evt_type);
    }
   }
   else if(pts.size() == 1u){
    dbl yl = k[i] * common_l + b[i], yl1 = k[j] * common_l +
         b[j];
    int evt_type = -segtype[i] * segtype[j];
    if(lt(yl1, yl) == (segtype[i] == 1)){
     evts.emplace_back(common_l, evt_type);
     evts.emplace_back(pts[0].x, -evt_type);
    }
    yl = k[i] * common_r + b[i], yl1 = k[j] * common_r + b[j];
    if(lt(yl1, yl) == (segtype[i] == 1)){
     evts.emplace_back(pts[0].x, evt_type);
     evts.emplace_back(common_r, -evt_type);
    }
   }
   else{
    if(segtype[j] != segtype[i] || j > i){
     evts.emplace_back(common_l, -2);
     evts.emplace_back(common_r, 2);
    }
   }
  }
  evts.emplace_back(l, 0);
  sort(evts.begin(), evts.end());
  size_t j = 0;
  int balance = 0;
  while(j < evts.size()){
   size_t ptr = j;
   while(ptr < evts.size() && eq(evts[j].first,
        evts[ptr].first)){
    balance += evts[ptr].second;
    ++ptr;
   }
   if(!balance && !eq(evts[j].first, r)){
    dbl next_x = ptr == evts.size() ? r : evts[ptr].first;
    ans -= segtype[i] * (k[i] * (next_x + evts[j].first) + 2 *
         b[i]) * (next_x - evts[j].first);
   }
   j = ptr;
  }
 }
 return ans/2;
}
```

76 Voronoi

```
const Tf INF = 1e10;
vector<Polygon> voronoi(vector<PT> site, Tf bsq) {
  int n = site.size();
  vector<Polygon> region(n);
  PT A(-bsq, -bsq), B(bsq, -bsq),
  C(bsq, bsq), D(-bsq, bsq);
  for(int i = 0; i < n; ++i) {
    vector<DirLine> li(n - 1);
    for(int j = 0, k = 0; j < n; ++j) {
      if(i == j) continue;
      li[k++] = DirLine((site[i] + site[j]) / 2,
      rotate90(site[j] - site[i]));
    }
    li.emplace_back(A,B-A); li.emplace_back(B,C-B);
    li.emplace_back(C,D-C); li.emplace_back(D,A-D);
    region[i] = halfPlaneIntersection(li);
  }
  return region;
}
```

77 XORTrick

```
vector<int> basis[N];
int sz[N],a[N],LOGK = 21;
void insert_vector(vector<int>&basis ,int &sz, int mask){
 for(int i=0;i<LOGK;i++) {
  if(!(mask&(1<<i))) continue;
  if(!basis[i]) {
   basis[i] = mask; sz++; return;
  }
  mask^=basis[i];
 }
}
bool check(vector<int>&basis , int mask){
 for(int i=0;i<LOGK;i++) {
  if(!(mask&(1<<i))) continue;
  if(!basis[i]) { return 0; }
  mask ^= basis[i];
 }
 return 1;
}
```