Contents

1 Data Structures

1.1 BIT

```
//1-based
void update(int n, int idx, int v){
  while(id <= n) tree[idx] += v, idx += idx & (-idx);
}
int query(int id){
  int sum = 0;
  while(idx > 0) sum += tree[idx], idx -= idx & (-idx);
  return sum;
}
```

1.2 CHT

```
/*
If m is decreasing:
  for min : bad(s-3, s-2, s-1), for max : bad(s-1, s-2, s-3)
If m is increasing:
  for max : bad(s-3, s-2, s-1), for min : bad(s-1, s-2, s-3)
If x isn't monotonic, then do Ternary Search or keep intersections and do binary search
*/
struct CHT{
  vector<ll> m, b;
  int ptr = 0;
  bool bad(int l1, int l2, int l3) { // returns intersect(l1, l3) <= intersect(l1, l2)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <= 1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]);
  }
  void insert_line(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while(s >= 3 && bad(s-3, s-2, s-1)) {
      s--;
      m.erase(m.end()-2);
      b.erase(b.end()-2);
    }
  }
  ll f(int i, ll x) { return m[i]*x + b[i]; }
  ll eval(ll x) {
    if(ptr >= m.size()) ptr = m.size()-1;
    while(ptr < m.size()-1 && f(ptr+1, x) > f(ptr, x)) ptr++;
    return f(ptr, x);
  }
};
```

1.3 Centroid Decomposition

```
const int MAX = 1e5 + 5;
int nn; bool isCentroid[MAX]; int sub[MAX]; // internal, won't need anymore
vector<int> ed[MAX]; // original graph adjacency list
int cpar[MAX]; // parent node in CD graph
int clevel[MAX]; // level in CD graph
int dis[20][MAX]; // distance of node from the centroid at i'th level
void calcSubTree(int s, int p) {
 sub[s] = 1;
 for(int x : ed[s]) {
  if(x == p or isCentroid[x]) continue;
  calcSubTree(x, s);
  sub[s] += sub[x];
 }
}
int getCentroid(int s, int p) {
 for(int x : ed[s]) {
  if(!isCentroid[x] && x != p && sub[x] > (nn / 2))
    return getCentroid(x, s);
 }
 return s;
}
void setDis(int s, int from, int p, int lev) {
 dis[from][s] = lev;
 for(int x : ed[s]) {
  if(x == p or isCentroid[x]) continue;
  setDis(x, from, s, lev + 1);
 }
}
void decompose(int s, int p, int lev) {
 calcSubTree(s, p); nn = sub[s];
 int c = getCentroid(s, p);
 setDis(c, lev, p, 0);
 /*
  for offline setDis() not needed,
  query() a child of c, add() that
  child to the global ds, reverse the
  edge list and do the same thing
  again, query and add are two dfs
  basically
 */
 isCentroid[c] = true; cpar[c] = p; clevel[c] = lev;
 for(int x : ed[c]) {
  if(!isCentroid[x]) decompose(x, c, lev + 1);
 }
}
// nn = n; decompose(1, -1, 0)
```

1.4 Dynamic CHT

```
const ll is_query = -LLONG_MAX;
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    //may need to use __int128 instead of ld if supported
    return ld(x->b - y->b)*(z->m - y->m) >= ld(y->b - z->b)*(y->m - x->m);
  }
  void insert_line(ll m, ll b) {
    auto y = insert({ -m, -b }); //change here for min
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    while (y != begin() && bad(prev(y))) erase(prev(y));
    if(y != begin()) prev(y)->succ = [=] { return &*y; };
  }
  ll eval(ll x) {
    auto l = *lower_bound((Line) { x, is_query });
    return -(l.m * x + l.b); //change here for min
  }
} hull;
```

1.5 HLD

```
/*
subtree of v corresponds to segment [in[v], out[v])
   and the path from v to the last vertex in
   ascending heavy path from v (which is nxt[v])
   will be [in[nxt[v]], in[v]]
*/
const int N = 1e5 + 5;
const int LOGN = 18;
int par[N], nxt[N], in[N], out[N], sz[N], h[N];
int n, timer;
vector<int> g[N];
void dfs_sz(int u = 0, int p = -1, int d = 0){
  par[u] = p, sz[u] = 1, h[u] = d;
  for(auto &v : g[u]){
    if(v ^ p) {
      dfs_sz(v, u, d + 1);
      sz[u] += sz[v];
      if(sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
  }
}
void dfs_hld(int u = 0, int p = -1){
  update(1, 0, n - 1, timer, val[u]);
  in[u] = timer++;
  for(auto v : g[u]){
    if(v ^ p){
      nxt[v] = (v == g[u][0]) ? nxt[u] : v;
      dfs_hld(v, u);
    }
  }
  out[u] = timer;
}
int hld_query(int u, int v){
  int ret = 0;
  while(nxt[u] != nxt[v]){
    if(h[nxt[u]] > h[nxt[v]]) swap(u, v);
    ret = merge(ret, query(1, 0, n - 1, in[nxt[v]], in[v]));
    v = par[nxt[v]];
```

```
  }
  if(h[u] > h[v]) swap(u, v);
  //in[u] -> in[u] + 1 in case of edge values
  ret = merge(ret, query(1, 0, n - 1, in[u],
    in[v]));
  return ret;
}
```

1.6 LCA

```
const int N = 1e5 + 5;
const int LOGN = 18;
int h[N], table[LOGN][N];
std::vector<int> g[N];
void dfs(int u = 0, int p = -1, int d = 0){
  table[0][u] = p, h[u] = d;
  for(int i = 1; i < LOGN; i++){
    if(table[i - 1][u] ^ -1) {
      table[i][u] = table[i - 1][table[i - 1][u]];
    }
    else table[i][u] = -1;
  }
  for(auto v : g[u]){
    if(v ^ p) dfs(v, u, d + 1);
  }
}
int get_lca(int u, int v) {
  if (h[u] < h[v]) swap(u, v);
  for (int i = LOGN - 1; i >= 0; i--) {
    if (h[u] - (1 << i) >= h[v]) u = table[i][u];
  }
  if (u == v) return u;
  for (int i = LOGN - 1; i >= 0; i--) {
    if (table[i][u] != table[i][v]) {
      u = table[i][u]; v = table[i][v];
    }
  }
  return table[0][u];
}
```

1.7 LIS

```
const int inf = 1e9;
vector<int> LIS(vector<int> a, int n){
  vector<int> d(n + 1, inf);
 for (int i = 0; i < n; i++) {
  *lower_bound(d.begin(), d.end(), a[i]) = a[i];
 }
  d.resize(lower_bound(d.begin(), d.end(), inf) -
    d.begin());
  return d;
}
```

1.8 Li Chao Tree

```
//Li Chao Tree for minimum case
struct Line {
  ll m, c;
  Line(ll m = 0, ll c = 0) : m(m), c(c) {};
  inline ll f(ll x) { return m * x + c; }
};
Line tree[4 * N];
void insert(int rt, int l, int r, Line v){
  if(l == r){
    if(tree[rt].f(l) > v.f(l)) tree[rt] = v;
    //change to < for max
    return;
  }
  int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
  bool lft = v.f(l) < tree[rt].f(l); //change to >
    for max
  bool mid = v.f(m) < tree[rt].f(m); //change to >
    for max
  if(mid) swap(tree[rt], v);
  if(lft != mid) insert(lc, l, m, v);
  else insert(rc, m + 1, r, v);
}
ll query(int rt, int l, int r, int x){
  if(l == r) return tree[rt].f(x);
  int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
  //replace min with max for max query
  if(x <= m) return min(tree[rt].f(x), query(lc, l,
    m, x));
  else return min(tree[rt].f(x), query(rc, m + 1,
    r, x));
}
```

1.9 Linear Sieve

```
const int PRIME_SZ = ;
int prime[PRIME_SZ], prime_sz;
bitset<N> mark;
void sieve(){
  for(int i = 2; i < N; ++i){
    if(!mark[i])prime[prime_sz++] = i;
    for(int j = 0; j < prime_sz and i * prime[j] <
      N and (!j or j and i % prime[j - 1]); ++j){
      mark[i * prime[j]] = true;
    }
  }
}
```

1.10 MO's Algorithm

```
int curL = 0, curR = -1;
for(int i = 0; i < Q.sz; i++){
  while(curL > Q[i].L){
    curL--; add(curL);
  }
  while(curR < Q[i].R){
    curR++; add(curR);
  }
  while(curL < Q[i].L){
    remove(curL); curL++;
  }
  while(curR > Q[i].R){
    remove(curR); curR--;
  }
}
```

1.11 Matrix Expo

```
struct matrix {
  ll mat[100][100]; // make this as small as
    possible
  int dim;
  matrix(){};
  matrix(int d){
    dim = d;
    for(int i = 0; i < dim; i++)
      for(int j = 0; j < dim; j++) mat[i][j] = 0;
  }
  matrix operator *(const matrix &mul){
    matrix ret = matrix(dim);
    for(int i = 0; i < dim; i++){
      for(int j = 0; j < dim; j++){
        for(int k = 0; k < dim; k++){
          ret.mat[i][j] += mat[i][k] *
    mul.mat[k][j];
          ret.mat[i][j] %= MOD;
        }
      }
    }
    return ret;
  }
  matrix operator + (const matrix &add){
    matrix ret = matrix(dim);
    for(int i = 0; i < dim; i++){
      for(int j = 0; j < dim; j++){
        ret.mat[i][j] = mat[i][j] + add.mat[i][j];
        ret.mat[i][j] %= MOD;
      }
    }
    return ret ;
  }
  matrix operator ^(int p){
    matrix ret = matrix(dim);
    matrix m = *this;
    for(int i = 0;i < dim; i++) ret.mat[i][i] = 1;
    while(p){
      if(p & 1) ret = ret * m;
      m = m * m; p >>= 1;
    }
    return ret;
  }
};
```

1.12 Persistent Segment Tree

```
const int N = 1e5 + 5;
using ll = long long;
struct node {
  ll sum;
  int L, R;
  node(ll sum = 0, int L = -1, int R = -1){
    sum = sum, L = L, R = R;
  }
} t[N * 25];
int root[N];
int node_cnt;
void build(int rt, int l, int r){
  if(l == r) return void(t[rt] = node(0));
  int m = l + r >> 1;
  t[rt].L = ++node_cnt; t[rt].R = ++node_cnt;
  build(t[rt].L, l, m); build(t[rt].R, m + 1, r);
}
void update(int &rt, int l, int r, int pos, ll v){
  t[++node_cnt] = t[rt];
  t[rt = node_cnt].sum += v;
  if(l == r) return;
  int m = l + r >> 1;
  if(pos <= m) update(t[rt].L, l, m, pos, v);
```

```cpp
  else update(t[rt].R, m + 1, r, pos, v);
}
// a = root[l - 1], b = root[r]
int lesscnt(int a, int b, int l, int r, int k){
  if(r <= k) return t[b].sum - t[a].sum;
  int m = l + r >> 1;
  if(k <= m) return lesscnt(t[a].L, t[b].L, l, m, k);
  return lesscnt(t[a].L, t[b].L, l, m, k) +
         lesscnt(t[a].R, t[b].R, m + 1, r, k);
}
int kthnum(int a, int b, int l, int r, int k){
  if(l == r) return l;
  int m = l + r >> 1, lval = t[t[b].L].sum - t[t[a].L].sum;
  if(lval >= k) return kthnum(t[a].L, t[b].L, l, m, k);
  return kthnum(t[a].R, t[b].R, m + 1, r, k - lval);
}
int main() {
  //initialize : build with root[0]
  root[0] = ++node_cnt;
  build(root[0], 1, n);
  //When need to update -
  root[i] = root[i - 1];
  update(root[i]...);
}
```

1.13 Trie

```cpp
const int MAX = 1e5 + 5, ALPHA = 10; // total characters, alphabet size
const char START = '0'; // first letter in alphabet
inline scale(char c){ return c - START; }
struct Trie {
  int root, nodes, nxt[MAX][ALPHA], finished[MAX];
  Trie(){
    root = nodes = 1;
    memset(nxt, 0, sizeof nxt);
  }
  void insert(string s){
    int cur = root;
    for(auto c : s){
      if(!nxt[cur][scale(c)]) {
        nxt[cur][scale(c)] = ++nodes;
      }
      cur = nxt[cur][scale(c)];
    }
    finished[cur]++;
  }
  bool find(string s){
    int cur = root;
    for(auto c : s){
      if(!nxt[cur][scale(c)]) return false;
      cur = nxt[cur][scale(c)];
    }
    return finished[cur];
  }
  void erase(string s){ // may need to call find() before
    int cur = root;
    for(auto c : s) cur = nxt[cur][scale(c)];
    finished[cur]--;
  }
};
```

2 Geometry

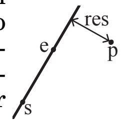
2.1 2D Primitive

2.1.1 Angle

A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

```cpp
/* Usage:
 *  vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
 *  int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
 *  // sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i
*/
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) < make_tuple(b.t, b.half(), a.x * (ll)b.y);
}
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ? make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

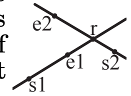
2.1.2 Line Distance



Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

```cpp
#include "Point.h"
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

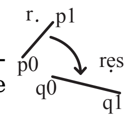
2.1.3 Line Intersection



If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

```cpp
/* Usage:
 *  auto res = lineInter(s1,e1,s2,e2);
 *  if (res.first == 1)
 *    cout << "intersection point at " << res.second << endl;
 */
#pragma once
#include "Point.h"
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

2.1.4 Linear Transformation



Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

```cpp
#include "Point.h"
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1, const P& q0, const P& q1, const P& r) {
```

```cpp
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

2.1.5 On Segment

```cpp
/* Description: Returns true iff p lies on the line segment from s to e.
 * Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.
 */
#include "Point.h"
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

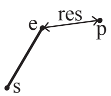
2.1.6 Point Sort

```cpp
// sort the points in counterclockwise order that starts from the half line x0,y=0.
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair <ll, ll> point;
#define x first
#define y second
int main() {
  int n; cin >> n;
  vector <point> p(n);
  for (auto &it : p) scanf("%lld %lld", &it.x, &it.y);
  sort(p.begin(), p.end(), [] (point a, point b) {
    return atan2l(a.y, a.x) < atan2l(b.y, b.x);
  });
  for (auto it : p) printf("%lld %lld\n", it.x, it.y);
  return 0;
}
```

2.1.7 Point

```cpp
// Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
 typedef Point P;
 T x, y;
 explicit Point(T x=0, T y=0) : x(x), y(y) {}
 bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
 bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
 P operator+(P p) const { return P(x+p.x, y+p.y); }
 P operator-(P p) const { return P(x-p.x, y-p.y); }
 P operator*(T d) const { return P(x*d, y*d); }
 P operator/(T d) const { return P(x/d, y/d); }
 T dot(P p) const { return x*p.x + y*p.y; }
 T cross(P p) const { return x*p.y - y*p.x; }
 T cross(P a, P b) const { return (a-*this).cross(b-*this); }
 T dist2() const { return x*x + y*y; }
 double dist() const { return sqrt((double)dist2()); }
 // angle to x-axis in interval [-pi, pi]
 double angle() const { return atan2(y, x); }
 P unit() const { return *this/dist(); } // makes dist()=1
 P perp() const { return P(-y, x); } // rotates +90 degrees
 P normal() const { return perp().unit(); }
 // returns point rotated 'a' radians ccw around the origin
 P rotate(double a) const {
  return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
 friend ostream& operator<<(ostream& os, P p) {
  return os << "(" << p.x << "," << p.y << ")"; }
};
```

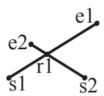
2.1.8 Segment Distance



Returns the shortest distance between point p and the line segment from point s to e.

```cpp
/* Usage:
 *  Point<double> a, b(2,2), p(1,1);
 *  bool onSegment = segDist(a,b,p) < 1e-10;
 * Status: tested
 */
#pragma once
#include "Point.h"
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

2.1.9 Segment Intersection



If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

```cpp
/* Usage:
 * vector<P> inter = segInter(s1,e1,s2,e2);
 * if (sz(inter)==1)
 *   cout << "segments intersect at " << inter[0] << endl;
 * Status: stress-tested, tested on kattis:intersection
 */
#include "Point.h"
#include "OnSegment.h"
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

2.1.10 Side Of

Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

```cpp
/* Usage:
 *  bool left = sideOf(p1,p2,q)==1;
 * Status: tested
 */
#include "Point.h"
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

2.2 3D

2.2.1 3D Convex Hull

```cpp
#include <bits/stdc++.h>
#define ll long long
#define sz(x) ((int) (x).size())
#define all(x) (x).begin(), (x).end()
#define vi vector<int>
#define pii pair<int, int>
#define rep(i, a, b) for(int i = (a); i < (b); i++)
using namespace std;
template<typename T>
using minpq = priority_queue<T, vector<T>, greater<T>>;
typedef long double ftype;
struct pt3 {
```

```
  ftype x, y, z;
  pt3(ftype x = 0, ftype y = 0, ftype z = 0) :
    x(x), y(y), z(z) {}
  pt3 operator-(const pt3 &o) const {
    return pt3(x - o.x, y - o.y, z - o.z);
  }
  pt3 cross(const pt3 &o) const {
    return pt3(y * o.z - z * o.y, z * o.x - x *
    o.z, x * o.y - y * o.x);
  }
  ftype dot(const pt3 &o) const {
    return x * o.x + y * o.y + z * o.z;
  }
};
// A face is represented by the indices of its
   three points a, b, c.
// It also stores an outward-facing normal vector q
struct face {
  int a, b, c;
  pt3 q;
};
// modify this depending on the coordinate sizes in
   your use case
const ftype EPS = 1e-9;
vector<face> hull3(const vector<pt3> &p) {
  int n = sz(p);
  assert(n >= 3);
  vector<face> f;
  // Consider an edge (a->b) dead if it is not a
     CCW edge of some current face
  // If an edge is alive but not its reverse, this
     is an exposed edge.
  // We should add new faces on the exposed edges.
  vector<vector<bool>> dead(n, vector<bool>(n,
    true));
  auto add_face = [&](int a, int b, int c) {
    f.push_back({a, b, c, (p[b] - p[a]).cross(p[c]
    - p[a])});
    dead[a][b] = dead[b][c] = dead[c][a] = false;
  };
  // Initialize the convex hull of the first 3
     points as a
  // triangular disk with two faces of opposite
     orientation
  add_face(0, 1, 2);
  add_face(0, 2, 1);
  rep(i, 3, n) {
    // f2 will be the list of faces invisible to
       the added point p[i]
    vector<face> f2;
    for(face &F : f) {
      if((p[i] - p[F.a]).dot(F.q) > EPS) {
        // this face is visible to the new point,
           so mark its edges as dead
        dead[F.a][F.b] = dead[F.b][F.c] =
        dead[F.c][F.a] = true;
      }else {
        f2.push_back(F);
      }
    }
    // Add a new face for each exposed edge.
    // Only check edges of alive faces for being
       exposed.
    f.clear();
    for(face &F : f2) {
      int arr[3] = {F.a, F.b, F.c};
      rep(j, 0, 3) {
        int a = arr[j], b = arr[(j + 1) % 3];
        if(dead[b][a]) {
          add_face(b, a, i);
        }
      }
    }
    f.insert(f.end(), all(f2));
  }
  return f;
}
```

2.2.2 Point3D

Class to handle points in 3D space.T can be e.g. double or long long.

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x),
    y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y,
    z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y,
    z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d);
    }
  P operator/(T d) const { return P(x/d, y/d, z/d);
    }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y -
    y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return
    sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in
    interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in
    interval [0, pi]
  double theta() const { return
    atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); }
    //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw
    around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u =
    axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

2.2.3 Polyhedron Volume

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

```
template<class V, class L>
double signedPolyVolume(const V& p, const L&
    trilist) {
  double v = 0;
  for (auto i : trilist) v +=
    p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

2.2.4 Spherical Distance

Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

2.3 Circle

2.3.1 Circle Intersection

Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

```
#include "Point.h"
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double
    r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 =
    r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() *
    sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

2.3.2 Circle Polygon Intersection

Returns the area of the intersection of a circle with a ccw polygon.
Time: $O(n)$

```
#include "Point.h"
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

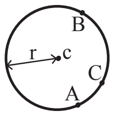
2.3.3 Circle Tangents

Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
#include "Point.h"
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

2.3.4 CircumCircle

The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.




```
#include "Point.h"
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

2.4 Polygon

2.4.1 Hull Diameter

Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

```
#include "Point.h"
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

2.4.2 Line Hull Intersection

Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:

$(-1,-1)$ if no collision,

$(i,-1)$ if touching the corner i,

(i,i) if along side $(i,i+1)$,

(i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.
Time: $O(\log n)$

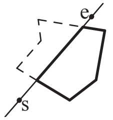
```
#include "Point.h"
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}
#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

2.4.3 Polygon Center

Returns the center of mass for a polygon.
Time: $O(n)$

```
#include "Point.h"
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

2.4.4 Polygon Cut



Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

```
/* Usage:
 *  vector<P> p = ...;
 *  p = polygonCut(p, P(0,0), P(1,0));
 * Status: tested but not extensively
 */
#include "Point.h"
#include "lineIntersection.h"
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

2.5 Closest Pair

Finds the closest pair of points.
Time: $O(n \log n)$

```
#include "Point.h"
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

2.6 Convex Hull

```
typedef long long ll;
typedef pair <ll, ll> point;
#define x first
#define y second
inline ll area (point a, point b, point c) {
  return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
vector <point> convexHull (vector <point> p) {
  int n = p.size(), m = 0;
  if (n < 3) return p;
  vector <point> hull(n + n);
  sort(p.begin(), p.end());
  for (int i = 0; i < n; ++i) {
    while (m > 1 and area(hull[m - 2], hull[m - 1], p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  for (int i = n - 2, j = m + 1; i >= 0; --i) {
    while (m >= j and area(hull[m - 2], hull[m - 1], p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  hull.resize(m - 1); return hull;
}
```

2.7 Minimum Enclosing Circle

```
// Expected runtime: O(n)
// Solves Gym 102299J
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef pair <ld, ld> point;
#define x first
#define y second
point operator + (const point &a, const point &b) {
  return point(a.x + b.x, a.y + b.y);
}
point operator - (const point &a, const point &b) {
  return point(a.x - b.x, a.y - b.y);
}
point operator * (const point &a, const ld &b) {
  return point(a.x * b, a.y * b);
}
point operator / (const point &a, const ld &b) {
  return point(a.x / b, a.y / b);
}
const ld EPS = 1e-8;
const ld INF = 1e20;
const ld PI = acosl(-1);
inline ld dist (point a, point b) {
  return hypotl(a.x - b.x, a.y - b.y);
}
inline ld sqDist (point a, point b) {
  return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
inline ld dot (point a, point b) {
  return a.x * b.x + a.y * b.y;
}
inline ld cross (point a, point b) {
  return a.x * b.y - a.y * b.x;
}
inline ld cross (point a, point b, point c) {
  return cross(b - a, c - a);
}
inline point perp (point a) {
  return point(-a.y, a.x);
}
// circle through 3 points
pair <point, ld> getCircle (point a, point b, point c) {
  pair <point, ld> ret;
  ld den = (ld) 2 * cross(a, b, c);
  ret.x.x = ((c.y - a.y) * (dot(b, b) - dot(a, a)) - (b.y - a.y) * (dot(c, c) - dot(a, a))) / den;
  ret.x.y = ((b.x - a.x) * (dot(c, c) - dot(a, a)) - (c.x - a.x) * (dot(b, b) - dot(a, a))) / den;
  ret.y = dist(ret.x, a);
  return ret;
}
pair <point, ld> minCircleAux (vector <point> &s, point a, point b, int n) {
  ld lo = -INF, hi = INF;
  for (int i = 0; i < n; ++i) {
    auto si = cross(b - a, s[i] - a);
    if (fabs(si) < EPS) continue;
    point m = getCircle(a, b, s[i]).x;
    auto cr = cross(b - a, m - a);
    si < 0 ? hi = min(hi, cr) : lo = max(lo, cr);
  }
  ld v = 0 < lo ? lo : hi < 0 ? hi : 0;
  point c = (a + b) * 0.5 + perp(b - a) * v / sqDist(a, b);
  return {c, sqDist(a, c)};
}
pair <point, ld> minCircle (vector <point> &s, point a, int n) {
  random_shuffle(s.begin(), s.begin() + n);
  point b = s[0], c = (a + b) * 0.5;
  ld r = sqDist(a, c);
  for (int i = 1; i < n; ++i) {
    if (sqDist(s[i], c) > r * (1 + EPS)) {
      tie(c, r) = n == s.size() ? minCircle(s, s[i], i) : minCircleAux(s, a, s[i], i);
    }
  }
  return {c, r};
}
pair <point, ld> minCircle (vector <point> s) {
  assert(!s.empty());
  if (s.size() == 1) return {s[0], 0};
  return minCircle(s, s[0], s.size());
}
int n; vector <point> p;
int main() {
  cin >> n;
  while (n--) {
    double x, y;
    scanf("%lf %lf", &x, &y);
    p.emplace_back(x, y);
  }
  pair <point, ld> circ = minCircle(p);
  printf("%0.12f %0.12f %0.12f\n", (double) circ.x.x, (double) circ.x.y, (double) (0.5 * circ.y));
  return 0;
}
```

2.8 Point In Polygon

```cpp
// Test if a point is inside a convex polygon in O(lg n) time
// Solves SPOJ INOROUT
typedef long long ll;
typedef pair <ll, ll> point;
#define x first
#define y second
struct segment {
  point P1, P2;
  segment () {}
  segment (point P1, point P2) : P1(P1), P2(P2) {}
};
inline ll ccw (point A, point B, point C) {
  return (B.x - A.x) * (C.y - A.y) - (C.x - A.x) * (B.y - A.y);
}
inline bool pointOnSegment (segment S, point P) {
  ll x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2 = S.P2.y;
  ll a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = a * c + b * d, len = c * c + d * d;
  if (x1 == x2 and y1 == y2) return x1 == x and y1 == y;
  if (dot < 0 or dot > len) return 0;
  return x1 * len + dot * c == x * len and y1 * len + dot * d == y * len;
}
const int M = 17;
const int N = 10010;
struct polygon {
  int n; // n > 1
  point p[N]; // clockwise order
  polygon () {}
  polygon (int _n, point *T) {
    n = _n;
    for (int i = 0; i < n; ++i) p[i] = T[i];
  }
  bool contains (point P, bool strictlyInside) {
    int lo = 1, hi = n - 1;
    while (lo < hi){
      int mid = lo + hi >> 1;
      if (ccw(p[0], P, p[mid]) > 0) lo = mid + 1;
      else hi = mid;
    }
    if (ccw(p[0], P, p[lo]) > 0) lo = 1;
    if (!strictlyInside and pointOnSegment(segment(p[0], p[n - 1]), P)) return 1;
    if (!strictlyInside and pointOnSegment(segment(p[lo], p[lo - 1]), P)) return 1;
    if (lo == 1 or ccw(p[0], P, p[n - 1]) == 0) return 0;
    return ccw(p[lo], P, p[lo - 1]) < 0;
  }
};
```

3 Graph

3.1 Articulation Point and Bridge

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
vector<int> g[N];
int vis[N], low[N], cut[N], now = 0, n, m;
void dfs(int u, int p) {
  low[u] = vis[u] = ++now; int ch = 0;
  for(int v : g[u]){
    if(v ^ p) {
      if(vis[v]) low[u] = min(low[u], vis[v]);
      else {
        ch++; dfs(v, u);
        low[u] = min(low[u], low[v]);
        if(p + 1 && low[v] >= vis[u]) cut[u] = 1;
        if(low[v] > vis[u]) {
          printf("Bridge %d -- %d\n", u, v);
        }
      }
    }
  } if(p == -1 && ch > 1) cut[u] = 1;
}
void ArticulationPointAndBridge() {
  memset(vis, 0, sizeof vis);
  memset(low, 0, sizeof low);
  memset(cut, 0, sizeof cut);
  now = 0;
  for(int i = 0; i < n; i++) {
    if(!vis[i]) dfs(i, -1);
  }
}
```

3.2 BCC

```cpp
#include <bits/stdc++.h>
using namespace std;
using pii = pair<int, int>;
const int N = 1e5 + 5;
struct BccVertex {
  int n, nBcc, step, root;
  int dfn[N], low[N];
  vector<int> E[N], ap;
  vector<pii> bcc[N];
  int top;
  pii stk[N];
  void init(int _n) {
    n = _n, nBcc = step = 0;
    for (int i = 0; i < n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].push_back(v); E[v].push_back(u);
  }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    int son = 0;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        son++;
        stk[top++] = {u,v};
        DFS(v,u);
        if (low[v] >= dfn[u]) {
          if(v != root) ap.push_back(v);
          do {
            assert(top > 0);
            bcc[nBcc].push_back(stk[--top]);
          } while (stk[top] != pii(u,v));
          nBcc++;
        }
        low[u] = min(low[u], low[v]);
      } else {
        if (dfn[v] < dfn[u]) stk[top++] = pii(u,v);
        low[u] = min(low[u],dfn[v]);
      }
    }
    if (u == root && son > 1) ap.push_back(u);
  }
  // return the edges of each bcc;
  vector<vector<pii>> solve() {
    vector<vector<pii>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    ap.clear();
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        root = i;
        DFS(i,i);
      }
    }
    for(int i = 0; i < nBcc; i++)
      res.push_back(bcc[i]);
    return res;
  }
}graph;
```

3.3 Bridge Component

```cpp
const int N = 1e5 + 5;
namespace Bridge {
  /* call addEdge to add edges, edge indices will be given [1, edgeId]
  nodes are 1-indexed, [1, n]
  created components are 1-indexed, [1, compId]
  */
  vector<pair<int, int>> adj[N];   // (edge-id, node)
  int visited[N];  // 0 - unvisited, 1 - visiting, 2 - visited
  int st[N], low[N], clk = 0;
  bool isBridge[N];
  int edgeId = 0;
  /// For bridge tree components
  int who[N], compId = 0;
  vector<int> stk;
  void dfs(int u, int parEdge) {
    visited[u] = 1;
    low[u] = st[u] = ++clk;
    stk.push_back(u);
    for (auto p : adj[u]) {
      int e = p.first, v = p.second;
      if (e == parEdge) continue;
      if (visited[v] == 1) {
        low[u] = min(low[u], st[v]);
      }
      else if(visited[v] == 0){
        dfs(v, e);
```

```
      low[u] = min(low[u], low[v]);
    }
  }
  visited[u] = 2;
  if(st[u] == low[u]){    // bridge / component
    found
    ++compId;
    int cur;
    do {
      cur = stk.back();
      stk.pop_back();
      who[cur] = compId;
    } while(cur != u);
    if(parEdge != -1) {
      isBridge[parEdge] = true;
    }
  }
}
void clearAll(int n){
  for(int i = 0; i <= n; i++) {
    adj[i].clear();
    visited[i] = st[i] = 0;
  }
  for(int i = 0; i <= edgeId; i++) isBridge[i] =
    false;
  clk = edgeId = compId = 0;
}
void findBridges(int n){
  for(int i = 1; i <= n; i++){
    if(visited[i] == 0) dfs(i, -1);
  }
}
void addEdge(int u, int v){
  edgeId++;
  adj[u].emplace_back(edgeId, v);
  adj[v].emplace_back(edgeId, u);
}
};
```

3.4 DSU On Tree

```
/// n log n
vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
           mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
    /// now (*cnt[v])[c] is the number of vertices
    in subtree of vertex v that has color c. You
    can answer the queries easily.
    /// note that in this step *vec[v] contains all
    of the subtree of vertex v.
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}
```

3.5 Dinic

```
// O(V^2 E), solves SPOJ FASTFLOW
struct edge {
  int u, v;
  ll cap, flow;
  edge () {}
  edge (int u, int v, ll cap) : u(u), v(v),
    cap(cap), flow(0) {}
};
struct Dinic {
  int N;
  vector <edge> E;
  vector <vector <int>> g;
  vector <int> d, pt;
  Dinic (int N) : N(N), E(0), g(N), d(N), pt(N) {}
  void AddEdge (int u, int v, ll cap) {
    if (u ^ v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }
  bool BFS (int S, int T) {
    queue <int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k : g[u]) {
        edge &e = E[k];
        if (e.flow < e.cap and d[e.v] > d[e.u] + 1)
          {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    } return d[T] != N + 1;
  }
  ll DFS (int u, int T, ll flow = -1) {
    if (u == T or flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      edge &e = E[g[u][i]];
      edge &oe = E[g[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 and amt > flow) amt = flow;
        if (ll pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    } return 0;
  }
  ll MaxFlow (int S, int T) {
    ll total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = DFS(S, T)) total += flow;
    }
    return total;
  }
};
```

3.6 Directed MST

```
// Find the directed minimum spanning tree whose
  root is the vertex S.
#include<bits/stdc++.h>
using namespace std;
struct RollbackUF {
  vector<int> e; vector<pair<int,int>> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x :
    find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool unite(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
using ll = long long;
struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l,
  a->r); }
pair<ll, vector<int>> dmst(int n, int r,
  vector<Edge>& g) {
```

```
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new
   Node{e});
  ll res = 0;
  vector<int> seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  for(int s = 0; s < n; ++s) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) { /// found cycle, contract
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.unite(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi],
   &Q[end]}});
      }
    }
    for(int i = 0; i < qi; ++i) in[uf.find(Q[i].b)]
   = Q[i];
  }
  for (auto& [u,t,comp] : cycs) { // restore sol
   (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  for(int i = 0; i < n; ++i) par[i] = in[i].a;
  return {res, par};
}
int main() {
  cout.tie(nullptr)->sync_with_stdio(false);
  int N, M, S; cin >> N >> M >> S;
  vector<Edge> edges(M);
  for (int i = 0; i < M; ++i) {
    int a, b, c; cin >> a >> b >> c;
    edges[i] = {a, b, c};
  }
  auto res = dmst(N, S, edges);
  cout << res.first << '\n';
  //print parent or root
  for (int i = 0; i < int(res.second.size()); ++i) {
    if (i == S) cout << S << ' ';
    else cout << res.second[i] << ' ';
  }
  cout << '\n';
}
```

3.7 Edmonds Blossom

```
//Maximum matching on general graph
struct Blossom {
  vector <int> q, g[N];
  int t, n, vis[N], par[N], orig[N], match[N],
   aux[N];
  Blossom (int _n = 0) {
    n = _n, t = 0;
    for (int i = 0; i <= n; ++i) {
      g[i].clear(), match[i] = aux[i] = par[i] = 0;
    }
  }
  inline void AddEdge (int u, int v) {
    g[u].push_back(v), g[v].push_back(u);
  }
  void augment (int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v], nv = match[pv];
      match[v] = pv, match[pv] = v, v = nv;
    } while (u ^ pv);
  }
  int lca (int u, int v) {
    ++t;
    while (true) {
      if (u) {
        if (aux[u] == t) return u; aux[u] = t;
        u = orig[par[match[u]]];
      }
      swap(u, v);
    }
  }
  void blossom (int u, int v, int x) {
    while (orig[u] ^ x) {
      par[u] = v, v = match[u];
      if (vis[v] == 1) q.emplace_back(v), vis[v] =
   0;
      orig[u] = orig[v] = x, u = par[v];
    }
  }
  bool bfs (int src) {
    fill(vis + 1, vis + n + 1, -1);
    iota(orig + 1, orig + n + 1, 1);
    q.clear(), q.emplace_back(src), vis[src] = 0;
    for (int i = 0; i < q.size(); ++i) {
      int u = q[i];
      for (int v : g[u]) {
        if (vis[v] == -1) {
          par[v] = u, vis[v] = 1;
          if (!match[v]) return augment(src, v), 1;
          q.emplace_back(match[v]), vis[match[v]] =
   0;
        } else if (vis[v] == 0 and orig[u] ^
   orig[v]) {
          int x = lca(orig[u], orig[v]);
          blossom(v, u, x), blossom(u, v, x);
        }
      }
    } return 0;
  }
  int maxMatch() {
    int ans = 0;
    vector <int> vec(n - 1);
    iota(vec.begin(), vec.end(), 1);
    shuffle(vec.begin(), vec.end(), mt19937(69));
    for (int u : vec) if (!match[u]) {
      for (int v : g[u]) if (!match[v]) {
        match[u] = v, match[v] = u;
        ++ans; break;
      }
    }
    for (int i = 1; i <= n; ++i) if (!match[i] and
   bfs(i)) ++ans;
    return ans;
  }
};
```

3.8 Eulerian Path

```
#include <bits/stdc++.h>
using namespace std;
// Eulerian path / circuit
// Undirected graph: circuit (or edge disjoint
   cycles) exists iff all nodes are of even degree
// Undirected graph: path exists iff number of odd
   degree nodes is zero or two
// Directed graph: circuit (or edge disjoint
   directed cycles) exists iff each node
//   satisfies in_degree = out_degree and the graph
   is strongly connected
// Directed graph: path exists iff at most one
   vertex has in_degree - out_degree = 1
//   and at most one vertex has out_degree -
   in_degree = 1 and all other vertices have
//   in_degree = out_degree, and graph is weakly
   connected
const int N = 200010;
bitset <N> bad;
vector <int> g[N];
vector <int> circ;
int n, m, deg[N], U[N], V[N];
void hierholzer (int src) {
  if (!deg[src]) return;
  vector <int> path;
  path.push_back(src);
  int at = src;
  while (!path.empty()) {
    if (deg[at]) {
      path.push_back(at);
      while (bad[g[at].back()]) g[at].pop_back();
      int e = g[at].back(), nxt = U[e] ^ at ^ V[e];
      bad[e] = 1, --deg[at], at = nxt; //change for
   directed
    } else {
      circ.push_back(at);
      at = path.back(), path.pop_back();
    }
  }
  reverse(circ.begin(), circ.end());
}
int main() {
  cin >> n >> m;
  for (int i = 1; i <= m; ++i) {
    scanf("%d %d", U + i, V + i);
    g[U[i]].push_back(i);
    g[V[i]].push_back(i); //change for directed
  }
  for(int i = 1; i <= n; i++) deg[i] = g[i].size();
  hierholzer(1); //change for directed [out(src) -
   in(src) = 1]
  for (int x : circ) printf("%d ", x); puts("");
  return 0;
}
```

3.9 Hopcroft Karp

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 40010;
const int INF = 1e8 + 5;
vector <int> g[N];
int n, m, p, match[N], dist[N];
bool bfs() {
  queue <int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : g[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1,
        q.emplace(match[v]);
      }
    }
  }
  return dist[0] != INF;
}
bool dfs (int u) {
  if (!u) return 1;
  for (int v : g[u]) {
    if (dist[match[v]] == dist[u] + 1 and dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  }
  dist[u] = INF;
  return 0;
}
int hopcroftKarp() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
      ret += !match[i] and dfs(i);
    }
  }
  return ret;
}
int main() {
  cin >> n >> m;
  // Bipartite Graph
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  // Maximum Matching, Minimum Vertex Cover
  int ans = hopcroftKarp();
  // Maximum Independent Set
  int offset = n - ans;
  cout << ans << " " << offset << '\n';
  return 0;
}
```

3.10 Hungarian

```cpp
// Given NN matrix A[i][j]. Calculate a permutation p[i] that minimize A[i][p[i]].
template <typename T>
pair <T, vector <int>> Hungarian (int n, int m, T c[N][N]) {
  vector <T> v(m), dist(m);
  vector <int> L(n, -1), R(m, -1);
  vector <int> index(m), prev(m);
  auto residue = [&] (int i, int j) {return c[i][j] - v[j];};
  iota(index.begin(), index.end(), 0);
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j), prev[j] = f;
    }
    T w; int i, j, l, s = 0, t = 0;
    while (true) {
      if (s == t) {
        l = s, w = dist[index[t++]];
        for (int k = t; k < m; ++k) {
          j = index[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) t = s, w = h;
            index[k] = index[t], index[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = index[k];
          if (R[j] < 0) goto augment;
        }
      }
      int q = index[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = index[k];
        T h = residue(i, j) - residue(i, q) + w;
        if (h < dist[j]) {
          dist[j] = h, prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto augment;
            index[k] = index[t], index[t++] = j;
          }
        }
      }
    }
  augment:
    for (int k = 0; k < l; ++k) v[index[k]] += dist[index[k]] - w;
    do {
      R[j] = i = prev[j], swap(j, L[i]);
    } while (i ^ f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) ret += c[i][L[i]];
  return {ret, L};
}
```

3.11 Kuhn

```cpp
int n, m;
vector<vector<int>> g; // [0..n) -> [0..m)
vector<int> mtn, mtm;
vector<char> used;
bool try_kuhn(int v) {
  if (used[v]) return false;
  used[v] = true;
  for (int to : g[v]) {
    if (mtm[to] == -1 or try_kuhn(mtm[to])) {
      mtm[to] = v;
      mtn[v] = to;
      return true;
    }
  }
  return false;
}
void match() {
  mtn.assign(n, -1);
  mtm.assign(m, -1);
  vector<int> order(n);
  iota(order.begin(), order.end(), 0);
  // modify order if custom order needed
  used.assign(n, false);
  for (int v : order) {
    if (mtn[v] == -1 and try_kuhn(v)) {
      used.assign(n, false);
    }
  }
}
int main() {
  // ... read the graph ...
  match();
  for (int i = 0; i < m; ++i) {
    if (mtm[i] != -1) {
      printf("%d %d\n", mtm[i] + 1, i + 1);
    }
  }
  return 0;
}
```

3.12 MCMF

```cpp
/***
 * 1 BASED NODE INDEXING
 * call init at the start of every test case
 * Complexity --> E*Flow (A lot less actually, not sure)
 * Maximizes the flow first, then minimizes the cost
 * The algorithm finds a path with minimum cost to send one unit of flow
     and sends flow over the path as much as possible. Then tries to find
     another path in the residual graph.
 * SPFA Technique :
       The basic idea of SPFA is the same as Bellman Ford algorithm in that each
       vertex is used as a candidate to relax its adjacent vertices. The improvement
       over the latter is that instead of trying all vertices blindly, SPFA maintains
       a queue of candidate vertices and adds a vertex to the queue only if that vertex
       is relaxed. This process repeats until no more vertex can be relaxed.
       This doesn't work if there is a negative cycle in the graph
 ***/
namespace mcmf {
```

```
using T = int;
const T INF = ?;     // 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
const int MAX = ?;     // maximum number of nodes
int n, src, snk;
T dis[MAX], mCap[MAX];
int par[MAX], pos[MAX];
bool vis[MAX];
struct Edge {
  int to, rev_pos;
  T cap, cost, flow;
};
vector<Edge> ed[MAX];
void init(int _n, int _src, int _snk) {
  n = _n, src = _src, snk = _snk;
  for (int i = 1; i <= n; i++) ed[i].clear();
}
void addEdge(int u, int v, T cap, T cost) {
  Edge a = {v, (int)ed[v].size(), cap, cost, 0};
  Edge b = {u, (int)ed[u].size(), 0, -cost, 0};
  ed[u].push_back(a);
  ed[v].push_back(b);
}
inline bool SPFA() {
  memset(vis, false, sizeof vis);
  for (int i = 1; i <= n; i++) mCap[i] = dis[i] = INF;
  queue<int> q;
  dis[src] = 0;
  vis[src] = true;       /// src is in the queue now
  q.push(src);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;        /// u is not in the queue now
    for (int i = 0; i < (int)ed[u].size(); i++) {
      Edge &e = ed[u][i];
      int v = e.to;
      if (e.cap > e.flow && dis[v] > dis[u] + e.cost) {
        dis[v] = dis[u] + e.cost;
        par[v] = u;
        pos[v] = i;
        mCap[v] = min(mCap[u], e.cap - e.flow);
        if (!vis[v]) {
          vis[v] = true;
          q.push(v);
        }
      }
    }
  }
  return (dis[snk] != INF);
}
inline pair<T, T> solve() {
  T F = 0, C = 0, f;
  int u, v;
  while (SPFA()) {
    u = snk;
    f = mCap[u];
    F += f;
    while (u != src) {
      v = par[u];
      ed[v][pos[u]].flow += f;          // edge of v-->u increases
      ed[u][ed[v][pos[u]].rev_pos].flow -= f;
      u = v;
    }
    C += dis[snk] * f;
  }
  return make_pair(F, C);
}
}
```

3.13 Manhattan MST

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct UnionFind {
  vector<int> UF; int cnt; UnionFind(int N) : UF(N, -1), cnt(N) {}
  int find(int v) { return UF[v] < 0 ? v : UF[v] = find(UF[v]); }
  bool join(int v, int w) {
    if ((v = find(v)) == (w = find(w))) return false;
    if (UF[v] > UF[w]) swap(v, w);
    UF[v] += UF[w]; UF[w] = v; cnt--; return true;
  }
  bool connected(int v, int w) { return find(v) == find(w); }
  int getSize(int v) { return -UF[find(v)]; }
};
template <class T> struct KruskalMST {
  using Edge = tuple<int, int, T>;
  T mstWeight; vector<Edge> mstEdges; UnionFind uf;
  KruskalMST(int V, vector<Edge> edges) : mstWeight(), uf(V) {
    sort(edges.begin(), edges.end(), [&] (const Edge &a, const Edge &b) {
      return get<2>(a) < get<2>(b);
    });
    for (auto &&e : edges) {
      if (int(mstEdges.size()) >= V - 1) break;
      if (uf.join(get<0>(e), get<1>(e))) {
        mstEdges.push_back(e); mstWeight += get<2>(e);
      }
    }
  }
};
template <class T> struct ManhattanMST : public KruskalMST<T> {
  using Edge = typename KruskalMST<T>::Edge;
  static vector<Edge> generateCandidates(vector<pair<T, T>> P) {
    vector<int> id(P.size()); iota(id.begin(), id.end(), 0); vector<Edge> ret;
    ret.reserve(P.size() * 4); for (int h = 0; h < 4; h++) {
      sort(id.begin(), id.end(), [&] (int i, int j) {
        return P[i].first - P[j].first < P[j].second - P[i].second;
      });
      map<T, int> M; for (int i : id) {
        auto it = M.lower_bound(-P[i].second);
        for (; it != M.end(); it = M.erase(it)) {
          int j = it->second;
          T dx = P[i].first - P[j].first, dy = P[i].second - P[j].second;
          if (dy > dx) break;
          ret.emplace_back(i, j, dx + dy);
        }
        M[-P[i].second] = i;
      }
      for (auto &&p : P) {
        if (h % 2) p.first = -p.first;
        else swap(p.first, p.second);
      }
    }
    return ret;
  }
  ManhattanMST(const vector<pair<T, T>> &P)
    : KruskalMST<T>(P.size(), generateCandidates(P)) {}
};
int main() {
  int N; cin >> N;
  vector<pair<ll, ll>> P(N);
  for (auto &&p : P) cin >> p.first >> p.second;
  ManhattanMST mst(P);
  cout << mst.mstWeight << '\n';
  for (auto &&[v, w, weight] : mst.mstEdges) cout << v << ' ' << w << '\n';
  return 0;
}
```

3.14 Maximum Independent Set

```
// maximum set of vertices in a graph, no two of which are adjacent.
struct MaxClique {
  // change to bitset for n > 64.
  static const int maxn = 64;
  int n, deg[maxn];
  uint64_t adj[maxn], ans;
  vector<pair<int, int>> edge;
  void init(int n_) {
    n = n_;
    fill(adj, adj + n, 0ull);
    fill(deg, deg + n, 0);
    edge.clear();
  }
  void add_edge(int u, int v) {
    edge.emplace_back(u, v);
    ++deg[u], ++deg[v];
  }
  vector<int> operator()() {
    vector<int> ord(n);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int u, int v) { return deg[u] < deg[v]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
      int u = id[e.first], v = id[e.second];
      adj[u] |= (1ull << v);
      adj[v] |= (1ull << u);
    }
    uint64_t r = 0, p = (1ull << n) - 1;
    ans = 0;
```

```
    dfs(r, p);
    vector<int> res;
    for (int i = 0; i < n; ++i) {
      if (ans >> i & 1) res.push_back(ord[i]);
    }
    return res;
  }
#define pcount __builtin_popcountll
  void dfs(uint64_t r, uint64_t p) {
    if (p == 0) {
      if (pcount(r) > pcount(ans)) ans = r;
      return;
    }
    if (pcount(r | p) <= pcount(ans)) return;
    int x = __builtin_ctzll(p & -p);
    uint64_t c = p;
    while (c > 0) {
      // bitset._Find_first(); bitset._Find_next();
      x = __builtin_ctzll(c & -c);
      r |= (1ull << x);
      dfs(r, p & adj[x]);
      r &= ~(1ull << x);
      p &= ~(1ull << x);
      c ^= (1ull << x);
    }
  }
};
```

3.15 SCC

```
// col[u] stores the component number u belongs to
vector<int> adj[N], trans[N];
int col[n], vis[n], idx = 0, n, m;
stack<int> st;
void dfs(int u) {
  vis[u] = 1;
  for(int v : adj[u]) if(!vis[v]) dfs(v);
  st.push(u);
}
void dfs2(int u) {
  col[u] = idx;
  for(int v : trans[u]) if(!col[v]) dfs2(v);
}
void scc() {
  for(int i = 1; i <= n; i++){
    if(!vis[i]) dfs(i);
  }
  while(!st.empty()) {
    int u = st.top(); st.pop();
    if(col[u]) continue;
    idx++; dfs2(u);
  }
}
```

3.16 TwoSat

```
/**
 * Description: Calculates a valid assignment to
 ↪  boolean variables a,
 * b, c,... to a 2-SAT problem, so that an
 ↪  expression of the type
 * $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$
 ↪  becomes true, or
 * reports that it is unsatisfiable. Negated
 ↪  variables are represented
 * by bit-inversions (\texttt{\tilde{}x}). Usage:
 ↪  TwoSat ts(number of
 * boolean variables); ts.either(0, \tilde3); //
 ↪  Var 0 is true or var
 * 3 is false ts.set_value(2); // Var 2 is true
 *  ts.at_most_one({0,\tilde1,2}); // <= 1 of vars
 ↪  0, \tilde1 and 2
 * are true ts.solve(); // Returns true iff it is
 ↪  solvable
 *  ts.values[0..N-1] holds the assigned values to
 ↪  the vars
 * Time: O(N+E), where N is the number of boolean
 ↪  variables, and E is
 * the number of clauses.
 */
struct TwoSat {
  int N;
  vector<vector<int>> gr;
  vector<int> values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), gr(2 * n) {}
  int add_var() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].push_back(j ^ 1);
    gr[j].push_back(f ^ 1);
  }
  void set_value(int x) { either(x, x); }
  void at_most_one(const vector<int> &li) { //
↪  (optional)
    if ((int)li.size() <= 1) return;
    int cur = ~li[0];
    for (int i = 2; i < (int)li.size(); i++) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vector<int> val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.push_back(i);
    for (auto &e : gr[i])
      if (!comp[e]) low = min(low, val[e] ?:
↪  dfs(e));
    if (low == val[i]) do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1) values[x >> 1] =
↪  x & 1;
      } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2 * N, 0);
    comp = val;
    for (int i = 0; i < 2 * N; i++) {
      if (!comp[i]) dfs(i);
    }
    for (int i = 0; i < N; i++) {
      if (comp[2 * i] == comp[2 * i + 1]) return 0;
    }
    return 1;
  }
};
```

4 Math

4.1 CRT

```
bool crt(long long k1, long long m1, long long k2,
 ↪  long long m2,
        long long &k, long long &m) {
  k1 %= m1;
  if (k1 < 0) k1 += m1;
  k2 %= m2;
  if (k2 < 0) k2 += m2;
  long long x, y, g;
  if (!diophantine(m1, -m2, k2 - k1, x, y, g)) {
    return false;
  }
  long long dx = m2 / g;
  long long delta = x / dx - (x % dx < 0);
  k = m1 * (x - dx * delta) + k1;
  m = m1 / g * m2;
  assert(0 <= k && k < m);
  return true;
}
```

4.2 Diophantine

```
template<typename T>
bool diophantine(T a, T b, T c, T &x, T &y, T &g) {
  if (a == 0 && b == 0) {
    if (c == 0) {
      x = y = g = 0;
      return true;
    }
    return false;
  }
  if (a == 0) {
    if (c % b == 0) {
      x = 0;
      y = c / b;
      g = abs(b);
      return true;
    }
    return false;
  }
  if (b == 0) {
    if (c % a == 0) {
      x = c / a;
      y = 0;
      g = abs(a);
      return true;
    }
    return false;
  }
  g = extgcd(a, b, x, y);
  if (c % g != 0) {
    return false;
  }
```

```
  T dx = c / a;
  c -= dx * a;
  T dy = c / b;
  c -= dy * b;
  x = dx + (T)((__int128)x * (c / g) % b);
  y = dy + (T)((__int128)y * (c / g) % a);
  g = abs(g);
  return true;
  // |x|, |y| <= max(|a|, |b|, |c|) [tested]
}
```

4.3 Discrete Log

```
// Returns minimum non-negative  st a^x = b (mod m)
   or -1 if doesn't exist
int discreteLog(int a, int b, int M) {
  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 = (1LL * k * a / g) % M;
  }
  int RT = sqrt(M) + 1, aRT = 1;
  for (int i = 0; i < RT; i++) aRT = (aRT * 1LL *
   a) % M;
  gp_hash_table<int, int> vals;
  for (int i = 0, cur = b; i <= RT; i++) {
    vals[cur] = i;
    cur = (cur * 1LL * a)%M;
  }
  for (int i = 1, cur = k; i <= M / RT + 1; i++) {
    cur = (cur * 1LL * aRT) % M;
    if (vals.find(cur) != vals.end())
      return RT * i - vals[cur] + add;
  }
  return -1;
}
```

4.4 Extended Euclid

```
// find x and y : ax + by = gcd(a, b)
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;
}
```

4.5 FFT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
struct cplx {
  ld a, b;
  cplx (ld a = 0, ld b = 0) : a(a), b(b) {}
  const cplx operator + (const cplx &c) const {
    return cplx(a + c.a, b + c.b);
  }
  const cplx operator - (const cplx &c) const {
    return cplx(a - c.a, b - c.b);
  }
  const cplx operator * (const cplx &c) const {
    return cplx(a * c.a - b * c.b, a * c.b + b *
   c.a);
  }
  const cplx conj() const {
    return cplx(a, -b);
  }
};
const ld PI = acosl(-1);
const int MOD = 1e9 + 7;
const int N = (1 << 20) + 5;
int rev[N]; cplx w[N];
void prepare (int &n) {
  int sz = __builtin_ctz(n);
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1]
   >> 1) | ((i & 1) << (sz - 1));
  w[0] = 0, w[1] = 1, sz = 1;
  while (1 << sz < n) {
    ld ang = 2 * PI / (1 << (sz + 1));
    cplx w_n = cplx(cosl(ang), sinl(ang));
    for (int i = 1 << (sz - 1); i < (1 << sz); ++i)
   {
      w[i << 1] = w[i], w[i << 1 | 1] = w[i] * w_n;
    } ++sz;
  }
}
void fft (cplx *a, int n) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int h = 1; h < n; h <<= 1) {
    for (int s = 0; s < n; s += h << 1) {
      for (int i = 0; i < h; ++i) {
        cplx &u = a[s + i], &v = a[s + i + h], t =
   v * w[h + i];
        v = u - t, u = u + t;
      }
    }
  }
}
static cplx f[N], g[N], u[N], v[N];
void multiply (int *a, int *b, int n, int m) {
  int sz = n + m - 1;
  while (sz & (sz - 1)) sz = (sz | (sz - 1)) + 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) f[i] = cplx(i < n ?
   a[i] : 0, i < m ? b[i] : 0);
  fft(f, sz);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (f[i] * f[i] - (f[j] * f[j]).conj()) *
   cplx(0, -0.25);
    f[j] = x, f[i] = x.conj();
  }
  fft(f, sz);
  for(int i = 0; i < sz; ++i) a[i] = f[i].a / sz +
   0.5;
}
inline void multiplyMod (int *a, int *b, int n, int
   m) {
  int sz = 1;
  while (sz < n + m - 1) sz <<= 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) {
    f[i] = i < n ? cplx(a[i] & 32767, a[i] >> 15) :
   cplx(0, 0);
    g[i] = i < m ? cplx(b[i] & 32767, b[i] >> 15) :
   cplx(0, 0);
  }
  fft(f, sz), fft(g, sz);
  for (int i = 0; i < sz; ++i) {
    int j = (sz - i) & (sz - 1);
    static cplx da, db, dc, dd;
    da = (f[i] + f[j].conj()) * cplx(0.5, 0);
    db = (f[i] - f[j].conj()) * cplx(0, -0.5);
    dc = (g[i] + g[j].conj()) * cplx(0.5, 0);
    dd = (g[i] - g[j].conj()) * cplx(0, -0.5);
    u[j] = da * dc + da * dd * cplx(0, 1);
    v[j] = db * dc + db * dd * cplx(0, 1);
  }
  fft(u, sz), fft(v, sz);
  for(int i = 0; i < sz; ++i) {
    int da = (ll) (u[i].a / sz + 0.5) % MOD;
    int db = (ll) (u[i].b / sz + 0.5) % MOD;
    int dc = (ll) (v[i].a / sz + 0.5) % MOD;
    int dd = (ll) (v[i].b / sz + 0.5) % MOD;
    a[i] = (da + ((ll) (db + dc) << 15) + ((ll) dd
   << 30)) % MOD;
  }
}
```

4.6 FWHT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1 << 20;
// apply modulo if necessary
void fwht_xor (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x + y, a[j + h] = x - y;
        if (dir) a[j] >>= 1, a[j + h] >>= 1;
      }
    }
  }
}
void fwht_or (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x, a[j + h] = dir ? y - x : x + y;
      }
    }
  }
}
void fwht_and (int *a, int n, int dir = 0) {
```

```cpp
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = dir ? x - y : x + y, a[j + h] = y;
      }
    }
  }
}
int n, a[N], b[N], c[N];
int main() {
  n = 1 << 16;
  for (int i = 0; i < n; ++i) {
    a[i] = rand() & 7;
    b[i] = rand() & 7;
  }
  fwht_xor(a, n), fwht_xor(b, n);
  for (int i = 0; i < n; ++i) {
    c[i] = a[i] * b[i];
  }
  fwht_xor(c, n, 1);
  for (int i = 0; i < n; ++i) {
    cout << c[i] << " ";
  }
  cout << '\n';
  return 0;
}
```

4.7 Floor Sum of AP

```cpp
ll sum(ll n){
  return n * (n - 1) >> 1;
}
// sum [(ai + b) / m] for 0 <= i < n
ll floorSumAP (ll a, ll b, ll m, ll n) {
  ll res = a / m * sum(n) + b / m * n;
  a %= m, b %= m; if (!a) return res;
  ll to = (n * a + b) / m;
  return res + (n - 1) * to - floorSumAP(m, m - 1 - b, a, to);
}
// sum (a + di) % m for 0 <= i < n
ll modSumAP (ll a, ll b, ll m, ll n) {
  b = ((b % m) + m) % m, a = ((a % m) + m) % m;
  return n * b + a * sum(n) - m * floorSumAP(a, b, m, n);
}
```

4.8 Gaussian Elimination

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 505;
const ld EPS = 1e-10;
const int MOD = 998244353;
ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}
pair <int, ld> gaussJordan (int n, int m, ld eq[N][N], ld res[N]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (fabs(eq[k][j]) > fabs(eq[piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x * eq[i][l];
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] * res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanModulo (int n, int m, int eq[N][N], int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] > eq[piv][j]) piv = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det = (ll) det * eq[i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) % mod;
      for (int l = j; l <= m; ++l) if (eq[i][l]) eq[k][l] = (eq[k][l] + mod_sq - x * eq[i][l]) % mod;
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m] * bigMod(eq[pos[i]][i], -1, mod) % mod;
  }
  for (int i = 0; i < n; ++i) {
    ll cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += (ll) eq[i][j] * res[j], cur %= mod;
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanBit (int n, int m, bitset <N> eq[N], bitset <N> &res) {
  int det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j]) {
      piv = k; break;
    }
    if (!eq[piv][j]) continue; pos[j] = i, swap(eq[piv], eq[i]), det &= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) eq[k] ^= eq[i]; ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m];
  }
  for (int i = 0; i < n; ++i) {
    int cur = eq[i][m];
    for (int j = 0; j < m; ++j) cur ^= eq[i][j] & res[j];
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
```

4.9 Lagrange Interpolation

```cpp
#include <bits/stdc++.h>
using namespace std;
struct LagrangeInterpolation {
  int deg, mod;
  vector<int> inv, fx, fy, c;
  LagrangeInterpolation(vector<pair<int, int>> points, int mod) {
    deg = points.size() - 1;
    this->mod = mod;
    inv.resize(mod);
    inv[1] = 1;
    for (int i = 2; i < mod; i++) {
      inv[i] = mul(mod - mod / i, inv[mod % i]);
    }
    for (auto [x, y] : points) {
      fx.emplace_back(x);
      fy.emplace_back(y);
    }
    c.resize(deg + 1);
    for (int i = 0; i <= deg; i++) {
      c[i] = 1;
      for (int j = 0; j <= deg; j++) {
        if (j != i) {
          c[i] = mul(c[i], inv[abs(fx[i] - fx[j])]);
          if (fx[i] < fx[j]) c[i] = mod - c[i];
        }
```

```
      }
      c[i] = mul(c[i], fy[i]);
    }
  }
  inline int mul(int a, int b) {
    return (a * 1ll * b) % mod;
  }
  int eval(int x) {
    if (find(fx.begin(), fx.end(), x) != fx.end()) {
      return fy[find(fx.begin(), fx.end(), x) - fx.begin()];
    }
    int prod = 1;
    for (int i = 0; i <= deg; i++) {
      prod = mul(prod, x - fx[i]);
    }
    int ret = 0;
    for (int i = 0; i <= deg; i++) {
      int curr = mul(c[i], mul(prod, inv[abs(x - fx[i])]));
      if (x < fx[i]) curr = mod - curr;
      ret = (ret + curr) % mod;
    }
    return ret;
  }
};
// LagrangeInterpolation li(points, mod);
// li.eval(n)
```

4.10 NTT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int G = 3;
const int MOD = 998244353;
const int N = (1 << 20) + 5;
int rev[N], w[N], inv_n;
int bigMod (int a, int e, int mod) {
  if (e == -1) e = mod - 2;
  int ret = 1;
  while (e) {
    if (e & 1) ret = (ll) ret * a % mod;
    a = (ll) a * a % mod; e >>= 1;
  }
  return ret;
}
void prepare (int n) {
  int sz = abs(31 - __builtin_clz(n));
  int r = bigMod(G, (MOD - 1) / n, MOD);
  inv_n = bigMod(n, MOD - 2, MOD), w[0] = w[n] = 1;
  for (int i = 1; i < n; ++i) w[i] = (ll) w[i - 1] * r % MOD;
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
}
void ntt (int *a, int n, int dir) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int m = 2; m <= n; m <<= 1) {
    for (int i = 0; i < n; i += m) {
      for (int j = 0; j < (m >> 1); ++j) {
        int &u = a[i + j], &v = a[i + j + (m >> 1)];
        int t = (ll) v * w[dir ? n - n / m * j : n / m * j] % MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  } if (dir) for (int i = 0; i < n; ++i) a[i] = (ll) a[i] * inv_n % MOD;
}
int f_a[N], f_b[N];
vector <int> multiply (vector <int> a, vector <int> b) {
  int sz = 1, n = a.size(), m = b.size();
  while (sz < n + m - 1) sz <<= 1; prepare(sz);
  for (int i = 0; i < sz; ++i) f_a[i] = i < n ? a[i] : 0;
  for (int i = 0; i < sz; ++i) f_b[i] = i < m ? b[i] : 0;
  ntt(f_a, sz, 0); ntt(f_b, sz, 0);
  for (int i = 0; i < sz; ++i) f_a[i] = (ll) f_a[i] * f_b[i] % MOD;
  ntt(f_a, sz, 1); return vector <int> (f_a, f_a + n + m - 1);
}
// G = primitive_root(MOD)
int primitive_root (int p) {
  vector <int> factor;
  int tmp = p - 1;
  for (int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.emplace_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if (tmp != 1) factor.emplace_back(tmp);
  for (int root = 1; ; ++root) {
    bool flag = true;
    for (int i = 0; i < (int) factor.size(); ++i) {
      if (bigMod(root, (p - 1) / factor[i], p) == 1) {
        flag = false; break;
      }
    }
    if (flag) return root;
  }
}
int main() {
  // (x + 2)(x + 3) = x^2 + 5x + 6
  vector <int> a = {2, 1};
  vector <int> b = {3, 1};
  vector <int> c = multiply(a, b);
  for (int x : c) cout << x << " "; cout << endl;
  return 0;
}
```

4.11 Pollard Rho

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
namespace Rho {
  ull mul (ull a, ull b, ull mod) {
    ll ret = a * b - mod * (ull) (1.L / mod * a * b);
    return ret + mod * (ret < 0) - mod * (ret >= (ll) mod);
  }
  ull bigMod (ull a, ull e, ull mod) {
    ull ret = 1;
    while (e) {
      if (e & 1) ret = mul(ret, a, mod);
      a = mul(a, a, mod), e >>= 1;
    }
    return ret;
  }
  bool isPrime (ull n) {
    if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
    ull a[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    ull s = __builtin_ctzll(n - 1), d = n >> s;
    for (ull x : a) {
      ull p = bigMod(x % n, d, n), i = s;
      while (p != 1 and p != n - 1 and x % n and i--) p = mul(p, p, n);
      if (p != n - 1 and i != s) return 0;
    }
    return 1;
  }
  ull pollard (ull n) {
    auto f = [&] (ull x) {return mul(x, x, n) + 1;};
    ull x = 0, y = 0, t = 0, prod = 2, i = 1, q;
    while (t++ % 40 or __gcd(prod, n) == 1) {
      if (x == y) x = ++i, y = f(x);
      if ((q = mul(prod, max(x, y) - min(x, y), n))) prod = q;
      x = f(x), y = f(f(y));
    }
    return __gcd(prod, n);
  }
  vector <ull> factor (ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
  }
};
int t; ll n;
int main() {
  cin >> t;
  while (t--) {
    scanf("%lld", &n);
    vector <ull> facs = Rho::factor(n);
    sort(facs.begin(), facs.end());
    printf("%d", (int) facs.size());
    for (auto it : facs) printf(" %llu", it);
    puts("");
  }
  return 0;
}
```

4.12 Power Sum

```
ll ncr[N][N], S[N][N], B[N], d[N][N];
ll fact[N], inv[N];
ll powerSum(ll n, ll k) {
  ll ret = 0LL;
  for (int i = 0; i <= k + 1; ++i) {
    ret += d[k][i] * bigMod(n, i);
    ret %= MOD;
  }
  return ret;
}
void init() {
  fact[0] = 1;
  for (int i = 1; i < N; ++i) { fact[i] = (fact[i -
  ↪ 1] * i) % MOD; }
  inv[1] = 1;
  for (int i = 2; i < N; ++i) {
    inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;
  }
  S[0][0] = 1;
  for (int i = 1; i < N; ++i) {
    S[i][0] = 0;
    for (int j = 1; j <= i; ++j) {
      S[i][j] = j * S[i - 1][j] + S[i - 1][j - 1];
      S[i][j] %= MOD;
    }
  }
  for (int i = 1; i < N; ++i) {
    ncr[i][0] = ncr[i][i] = 1;
    for (int j = 1; j < i; ++j) {
      ncr[i][j] = ncr[i - 1][j] + ncr[i - 1][j - 1];
      ncr[i][j] %= MOD;
    }
  }
  for (int i = 0; i < 15; ++i) {
    for (int j = 0; j <= i; ++j) {
      B[i] += ((j & 1) ? -1 : 1) * ((fact[j] *
  ↪ S[i][j]) % MOD)
              * inv[j + 1];
      B[i] %= MOD;
    }
  }
  for (int i = 0; i < 15; ++i) {
    for (int j = 0; j <= i; ++j) {
      d[i][i + 1 - j] = (ncr[i + 1][j] * abs(B[j]))
  ↪ % MOD;
      d[i][i + 1 - j] *= inv[i + 1];
      d[i][i + 1 - j] %= MOD;
    }
  }
  d[0][0] = 1; // 0^0 = 1
}
```

4.13 SQRT Mod

```
int power(long long n, long long k, const int mod) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
// find sqrt(a) % p, i.e. find any x such that x^2
 ↪ = a (mod p)
// p is prime
// complexity: O(log^2 p) worst case, O(log p) on
 ↪ average
// Tonelli-Shanks algorithm
int SQRT(int a, int p) {
 a %= p; if (a < 0) a += p;
 if (a == 0) return 0;
 if (power(a, (p-1)/2, p) != 1) return -1; //
 ↪ solution does not exist
 if (p % 4 == 3) return power(a, (p+1)/4, p);
 int s = p - 1, n = 2;
 int r = 0, m;
 while (s % 2 == 0) ++r, s /= 2;
 // find a non-square mod p
 while (power(n, (p - 1) / 2, p) != p - 1) ++n;
 int x = power(a, (s + 1) / 2, p);
 int b = power(a, s, p), g = power(n, s, p);
 for (;; r = m) {
  int t = b;
  for (m = 0; m < r && t != 1; ++m) t = 1LL * t * t
 ↪ % p;
  if (m == 0) return x;
  int gs = power(g, 1LL << (r - m - 1), p);
  g = 1LL * gs * gs % p;
  x = 1LL * x * gs % p;
  b = 1LL * b * g % p;
 }
}
```

4.14 Stirling

```
/*
 * Number of permutations of n elements with k
 ↪ disjoint cycles
   = Str1(n,k) = (n-1) * Str1(n-1,k) + Str1(n-1,k-1)
 * n! = Sum(Str1(n,k)) (for all 0 <= k <= n).
 * Ways to partition n labelled objects into k
 ↪ unlabelled
   subsets = Str2(n,k) = k * Str2(n-1,k) +
 ↪ Str2(n-1,k-1)
 * Parity of Str2(n,k) : ( (n-k) & Floor((k-1)/2) )
 ↪ == 0)
 * Ways to partition n labelled objects into k
 ↪ unlabelled
   subsets, with each subset containing at least r
 ↪ elements :
   SR(n,k) = k * SR(n-1,k) + C(n-1,r-1) *
 ↪ SR(n-r,k-1)
 * Number of ways to partition n labelled objects
 ↪ 1,2,3, ... n
   into k non-empty subsets so that for any
 ↪ integers i and j in a
   given subset |i-j| >= d : Str2(n-d+1, k-d+1), n
 ↪ >= k >= d
 */
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<ll> 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] * invfct[i]) % mod;
  }
  NTT ntt(mod);
  ntt.multiply(a, b, res);
  return res[r];
}
```

4.15 Sum of Totient Function

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int N = 3e5 + 9, mod = 998244353;
template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other)
    const { int32_t c = this->value + other.value;
 ↪ return modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other)
    const { int32_t c = this->value - other.value;
 ↪ return modint<MOD>(c <    0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other)
    const { int32_t c = (int64_t)this->value *
    other.value % MOD; return modint<MOD>(c < 0 ? c
 ↪ + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD>
    other) { this->value += other.value; if
    (this->value >= MOD) this->value -= MOD; return
 ↪ *this; }
  inline modint<MOD> & operator -= (modint<MOD>
    other) { this->value -= other.value; if
    (this->value <    0) this->value += MOD; return
 ↪ *this; }
  inline modint<MOD> & operator *= (modint<MOD>
    other) { this->value = (int64_t)this->value *
    other.value % MOD; if (this->value < 0)
 ↪ this->value += MOD; return *this; }
```

```cpp
  inline modint<MOD> operator - () const { return
    modint<MOD>(this->value ? MOD - this->value :
    0); }
  modint<MOD> pow(uint64_t k) const { modint<MOD> x
    = *this, y = 1; for (; k; k >>= 1) { if (k & 1)
    y *= x; x *= x; } return y; }
  modint<MOD> inv() const { return pow(MOD - 2); }
    // MOD must be a prime
  inline modint<MOD> operator /  (modint<MOD>
    other) const { return *this *  other.inv(); }
  inline modint<MOD> operator /= (modint<MOD>
    other)        { return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const
    { return value == other.value; }
  inline bool operator != (modint<MOD> other) const
    { return value != other.value; }
  inline bool operator < (modint<MOD> other) const
    { return value < other.value; }
  inline bool operator > (modint<MOD> other) const
    { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator *
    (int64_t value, modint<MOD> n) { return
    modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator *
    (int32_t value, modint<MOD> n) { return
    modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >>
    (istream & in, modint<MOD> &n) { return in >>
    n.value; }
template <int32_t MOD> ostream & operator <<
    (ostream & out, modint<MOD> n) { return out <<
    n.value; }
using mint = modint<mod>;
/**
Prefix sum of multiplicative functions :
    p_f : the prefix sum of f (x) (1 <= x <= T).
    p_g : the prefix sum of g (x) (0 <= x <= N).
    p_c : the prefix sum of f * g (x) (0 <= x <= N).
    T : the threshold, generally should be n ^ (2 /
    3). for n = 1e9, T = 1e6
Magic Functions for different f(x)
For f(x) = phi(x): Id(x) = phi * I(x) i.e. p_c =
    prefix sum of Id(x), p_g = prefix sum of I(x).
    Here Id(n) = n, I(n) = 1
For f(x) = mu(x): e(x) = mu * I(x). Here e(x) = x
    == 1 ? 1 : 0
Complexity: O(n^(2/3))
**/
namespace Dirichlet {
  //solution for f(x) = phi(x)
  const int T = 1e7 + 9;
  long long phi[T];
  gp_hash_table<long long, mint> mp;
  mint dp[T], inv;
  int sz, spf[T], prime[T];
  void init() {
    memset(spf, 0, sizeof spf);
    phi[1] = 1; sz = 0;
    for (int i = 2; i < T; i++) {
      if (spf[i] == 0) phi[i] = i - 1, spf[i] = i,
    prime[sz++] = i;
      for (int j = 0; j < sz && i * prime[j] < T &&
    prime[j] <= spf[i]; j++) {
        spf[i * prime[j]] = prime[j];
        if (i % prime[j] == 0) phi[i * prime[j]] =
    phi[i] * prime[j];
        else phi[i * prime[j]] = phi[i] * (prime[j]
    - 1);
      }
    }
    dp[0] = 0;
    for(int i = 1; i < T; i++) dp[i] = dp[i - 1] +
    phi[i] % mod;
    inv = 1; // g(1)
  }
  mint p_c(long long n) {
    if (n % 2 == 0) return n / 2 % mod * ((n + 1) %
    mod) % mod;
    return (n + 1) / 2 % mod * (n % mod) % mod;
  }
  mint p_g(long long n) {
    return n % mod;
  }
  mint solve (long long x) {
    if (x < T) return dp[x];
    if (mp.find(x) != mp.end()) return mp[x];
    mint ans = 0;
    for (long long i = 2, last; i <= x; i = last +
    1) {
      last = x / (x / i);
      ans += solve (x / i) * (p_g(last) - p_g(i -
    1));
    }
    ans = p_c(x) - ans;
    ans /= inv;
    return mp[x] = ans;
  }
};
```

4.16 Xor Basis

```cpp
void tryGauss(int mask) {
 for (int i = 0; i < n; i++) {
  if ((mask & 1 << i) == 0) continue;
  if (!basis[i]) {
   basis[i] = mask;
   ++sz;
   break;
  }
  mask ^= basis[i];
 }
}
```

5 Misc

5.1 DC Optimization

```cpp
void compute(int L, int R, int optL, int optR){
  if(L > R) return;
  int M = L + R >> 1;
  pair<ll, int> best(1LL << 60, -1);
  for(int k = optL; k <= min(M, optR); k++){
    best = min(best, {dp[prv][k] + C[k + 1][M], k});
  }
  dp[now][M] = best.ff;
  compute(L, M - 1, optL, best.ss);
  compute(M + 1, R, best.ss, optR);
}
```

5.2 Generator Utilities

```cpp
static random_device rd;
static mt19937 rng(rd());
mt19937 rng(seq);
// const int BASE =
    uniform_int_distribution<int>(1, MOD - 1)(rng);
// uniform_int_distribution<int> uid(low, high);
// auto gen = bind(uid, rng);
// shuffle(v.begin(), v.end(), rng)
mt19937 rng(chrono::steady_clock::now().time_since_
    epoch().count());
i64 my_rand(i64 l, i64 r) {
    return uniform_int_distribution<i64>(l, r)
    (rng);
}
```

5.3 Misc

```cpp
// Pragmas
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx,avx2,fma")
// Custom Priority Queue
std::priority_queue< int, std::vector<int>,
    std::greater<int> > Q; // increasing
//gp hash table
    https://codeforces.com/blog/entry/60737
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::n
    ow().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^
    RANDOM; }
};
gp_hash_table<key, int, chash> table;
//bitset
BS._Find_first()
BS._Find_next(x) //Return first set bit after xth
    bit, x on failure
//Gray Code, G(0) = 000, G(1) = 001, G(2) = 011,
    G(3) = 010
inline int g(int n){ return n ^ (n >> 1); }
//Inverse Gray Code
int rev_g(int g) {
  int n = 0;
  for (; g; g >>= 1) n ^= g;
  return n;
}
/// Only for non-negative integers
/// Returns the immediate next number with same
    count of one bits, -1 on failure
long long hakmemItem175(long long n){
  if(!n) return -1;
  long long x = (n & -n);
  long long left = (x + n);
  long long right = ((n ^ left) / x) >> 2;
  long long res = (left | right);
```

```cpp
  return res;
}
/// Returns the immediate previous number with same count of one bits, -1 on failure
long long lol(long long n){
  if(n < 2) return -1;
  long long res = ~hakmemItem175(~n);
  return (!res) ? -1 : res;
}
//Gilbert Ordering for Mo's Algorithm
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
  if (pow == 0) {
    return 0;
  }
  int hpow = 1 << (pow-1);
  int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) :
          ((y < hpow) ? 1 : 2);
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  int64_t subSquareSize = int64_t(1) << (2*pow - 2);
  int64_t ans = seg * subSquareSize;
  int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
  return ans;
}
struct Query {
  int l, r, idx; // queries
  int64_t ord; // Gilbert order of a query
  // call query[i].calcOrder() to calculate the Gilbert orders
  inline void calcOrder() {
    ord = gilbertOrder(l, r, 21, 0);
  }
};
// sort the queries based on the Gilbert order
inline bool operator<(const Query &a, const Query &b) {
  return a.ord < b.ord;
}
```

5.4 Ordered Multiset

```cpp
#include <bits/stdtr1c++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

/// Treap supporting duplicating values in set
/// Maximum value of treap * ADD must fit in long long

struct Treap{ /// hash = 96814
  int len;
  const int ADD = 1000010;
  const int MAXVAL = 1000000010;
  tr1::unordered_map <long long, int> mp; /// Change to int if only int in treap
  tree<long long, null_type, less<long long>,
    rb_tree_tag, tree_order_statistics_node_update> T;
  Treap(){
    len = 0;
    T.clear(), mp.clear();
  }
  inline void clear(){
    len = 0;
    T.clear(), mp.clear();
  }
  inline void insert(long long x){
    len++, x += MAXVAL;
    int c = mp[x]++;
    T.insert((x * ADD) + c);
  }
  inline void erase(long long x){
    x += MAXVAL;
    int c = mp[x];
    if (c){
      c--, mp[x]--, len--;
      T.erase((x * ADD) + c);
    }
  }
  /// 1-based index, returns the K'th element in the treap, -1 if none exists
  inline long long kth(int k){
    if (k < 1 || k > len) return -1;
    auto it = T.find_by_order(--k);
    return ((*it) / ADD) - MAXVAL;
  }
  /// Count of value < x in treap
  inline int count(long long x){
    x += MAXVAL;
    int c = mp[--x];
    return (T.order_of_key((x * ADD) + c));
  }
  /// Number of elements in treap
  inline int size(){
    return len;
  }
};
```

5.5 Ordered Set

```cpp
#include <bits/stdtr1c++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type,
less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// .insert(x)
// .find_by_order(k) //kth element
// //position of 1st element >= x
// .order_of_key(x)
```

5.6 SOS DP

```cpp
int A[1 << N], F[1 << N];
for (int mask = 0; mask < (1 << N); mask++) F[mask] = A[mask];
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];
```

5.7 debug

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sim template < class c
#define ris return * this
#define dor > debug & operator <<
#define eni(x) sim > typename \
  enable_if<sizeof dud<c>(0) x 1, debug&>::type operator<<(c i) {
sim > struct rge { c b, e; };
sim > rge<c> range(c i, c j) { return rge<c>{i, j}; }
sim > auto dud(c* x) -> decltype(cerr << *x, 0);
sim > char dud(...);
struct debug {
~debug() { cerr << endl; }
eni(!=) cerr << boolalpha << i; ris; }
eni(==) ris << range(begin(i), end(i)); }
sim, class b dor(pair < b, c > d) {
  ris << "(" << d.first << ", " << d.second << ")";
}
sim dor(rge<c> d) {
  *this << "[";
  for (auto it = d.b; it != d.e; ++it)
    *this << (it != d.b ? ", " : "") << *it;
  ris << "]";
}
};
#define name(...) " [" << #__VA_ARGS__ ": " << (__VA_ARGS__) << "] "
```

6 String

6.1 Aho Corasick

```cpp
#include <bits/stdc++.h>
using namespace std;
struct AC {
  int N, P;
  int A = 26;
  vector<vector<int>> next;
  vector<int> link, out_link;
  vector<vector<int>> out;
  AC() : N(0), P(0) {
    node();
  }
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  inline int get(char c) {
    return c - 'a';
  }
  int add_pattern(const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
```

```
    }
    out[u].push_back(P);
    return P++;
  }
  void compute() {
    queue<int> q;
    for (q.push(0); !q.empty(); ) {
      int u = q.front();
      q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) {next[u][c] = next[link[u]][c];}
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] =
            out[link[v]].empty() ?
  ↪  out_link[link[v]] : link[v];
          q.push(v);
        }
      }
    }
  }
  int advance(int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)];
    return u;
  }
  void match(const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v]) {
        for (auto p : out[v]) cout << "match " << p
  ↪  << endl;
      }
    }
  }
};
// Don't forget to call compute()!
int main() {
  AC aho;
  int n;
  cin >> n;
  while (n--) {
    string s;
    cin >> s;
    aho.add_pattern(s);
  }
  aho.compute();
  string text;
  cin >> text;
  aho.match(text);
  return 0;
}
```

6.2 Hash

```
struct Hash {
  struct base {
    string s; int b, mod;
    vector<int> hash, p;
    void init(string &_s, int _b, int _mod) { // b
  ↪  > 26, prime.
      s = _s; b = _b, mod = _mod;
      hash.resize(s.size());
      p.resize(s.size());
      hash[0] = s[0] - 'A' + 1; p[0] = 1;
      for(int i = 1; i < s.size(); ++i) {
        hash[i] = (ll) hash[i - 1] * b % mod;
        hash[i] += s[i] - 'A' + 1;
        if(hash[i] >= mod) hash[i] -= mod;
        p[i] = (ll) p[i - 1] * b % mod;
      }
    }
    int get(int l, int r) {
      int ret = hash[r];
      if(l) ret -= (ll) hash[l - 1] * p[r - l + 1]
  ↪  % mod;
      if(ret < 0) ret += mod;
      return ret;
    }
  } h[2];
  void init(string &s) {
    h[0].init(s, 29, 1e9+7);
    h[1].init(s, 31, 1e9+9);
  }
  pair<int, int> get(int l, int r) {
    return { h[0].get(l, r), h[1].get(l, r) };
  }
} H;
```

6.3 KMP

```
const int N = 2e6 + 5;
int pi[N]; //maximum suffix that is also a prefix
void prefix_function(string s){
 for(int i = 1; i < s.length(); i++){
  int j = pi[i - 1];
  while(j > 0 and s[i] != s[j]) j = pi[j - 1];
  if(s[i] == s[j]) j++;
  pi[i] = j;
 }
}
```

6.4 Manacher

```
/**
 * Description: For each position in a string,
   computes p[0][i] = half length of longest even
   palindrome around pos i, p[1][i] = longest odd
 ↪  (half rounded down).
 * Time: O(N)
 */
typedef vector<int> vi;
array<vi, 2> manacher(const string& s) {
  int n = s.length();
  array<vi, 2> p = { vi(n + 1), vi(n)};
  for (int z = 0; z < 2; z++) for (int i = 0, l =
 ↪  0, r = 0; i < n; i++) {
    int t = r - i + !z;
    if (i < r) p[z][i] = min(t, p[z][l + t]);
    int L = i - p[z][i], R = i + p[z][i] - !z;
    while (L >= 1 && R + 1 < n && s[L - 1] == s[R +
 ↪  1])
      p[z][i]++, L--, R++;
    if (R > r) l = L, r = R;
  }
  return p;
}
```

6.5 Palindromic Tree

```
#include <bits/stdc++.h>
using namespace std;
const int A = 26;
const int N = 300010;
char s[N]; long long ans;
int last, ptr, nxt[N][A], link[N], len[N], occ[N];
void feed (int at) {
  while (s[at - len[last] - 1] != s[at]) last =
 ↪  link[last];
  int ch = s[at] - 'a', temp = link[last];
  while (s[at - len[temp] - 1] != s[at]) temp =
 ↪  link[temp];
  if (!nxt[last][ch]) {
    nxt[last][ch] = ++ptr, len[ptr] = len[last] + 2;
    link[ptr] = len[ptr] == 1 ? 2 : nxt[temp][ch];
  }
  last = nxt[last][ch], ++occ[last];
}
int main() {
  len[1] = -1, len[2] = 0, link[1] = link[2] = 1,
 ↪  last = ptr = 2;
  scanf("%s", s + 1);
  for (int i = 1, n = strlen(s + 1); i <= n; ++i)
 ↪  feed(i);
  for (int i = ptr; i > 2; --i) ans = max(ans,
 ↪  len[i] * 1LL * occ[i]), occ[link[i]] += occ[i];
  printf("%lld\n", ans);
  return 0;
}
```

6.6 Persistent Trie

```
/***
    Given an array of size n, each value in array
 ↪  can be expressed using
    20 bits.
    Query : L R K
    max(a_i ^ K) for L <= i <= R
***/
const int MAX = 200010; /// maximum size of array
const int B = 19; /// maximum number of bits in a
 ↪  value - 1
int root[MAX], ptr = 0;
struct node {
    int ara[2], sum;
    node() {}
} tree[ MAX * (B+1) ];
void insert(int prevnode, int &curRoot, int val) {
    curRoot = ++ptr;
    int curnode = curRoot;
    for(int i = B; i >= 0; i--) {
        bool bit = val & (1 << i);
        tree[curnode] = tree[prevnode];
        tree[curnode].ara[bit] = ++ptr;
        tree[curnode].sum += 1;
        prevnode = tree[prevnode].ara[bit];
        curnode = tree[curnode].ara[bit];
    }
    tree[curnode] = tree[prevnode];
    tree[curnode].sum += 1;
}
```

```cpp
int find_xor_max(int prevnode, int curnode, int x) {
    int ans = 0;
    for(int i = B; i >= 0; i--) {
        bool bit = x & (1 << i);
        if(tree[tree[curnode].ara[bit ^ 1]].sum > tree[tree[prevnode].ara[bit ^ 1]].sum) {
            curnode = tree[curnode].ara[bit ^ 1];
            prevnode = tree[prevnode].ara[bit ^ 1];
            ans = ans | (1 << i);
        }
        else {
            curnode = tree[curnode].ara[bit];
            prevnode = tree[prevnode].ara[bit];
        }
    }
    return ans;
}
void solve() {
    int n, q, L, R, K;
    cin >> n;
    for(int i=1;i<=n;i++) cin >> ara[i];
    for(int i=1;i<=q;i++) {
        cin >> L >> R >> K;
        cout << find_xor_max(root[L-1],root[R],K) << endl;
    }
}
```

6.7 Suffix Array

```cpp
// Everything is 0-indexed
char s[N]; // Suffix array will be built for this string
int SA[N], iSA[N]; // SA is the suffix array, iSA[i] stores the rank of the i'th suffix
int cnt[N], nxt[N]; // Internal stuff
bool bh[N], b2h[N]; // Internal stuff
int lcp[N]; // Stores lcp of SA[i] and SA[i + 1]; lcp[n - 1] = 0
int lcpSparse[LOGN][N]; // lcpSparse[i][j] = min(lcp[j], ..., lcp[j - 1 + (1 << i)])
void buildSA(int n) {
  for (int i = 0; i < n; i++) SA[i] = i;
  sort(SA, SA + n, [](int i, int j) { return s[i] < s[j]; });
  for (int i = 0; i < n; i++) {
    bh[i] = i == 0 || s[SA[i]] != s[SA[i - 1]];
    b2h[i] = 0;
  }
  for (int h = 1; h < n; h <<= 1) {
    int tot = 0;
    for (int i = 0, j; i < n; i = j) {
      j = i + 1;
      while (j < n && !bh[j]) j++;
      nxt[i] = j; tot++;
    } if (tot == n) break;
    for (int i = 0; i < n; i = nxt[i]) {
      for (int j = i; j < nxt[i]; j++) iSA[SA[j]] = i;
      cnt[i] = 0;
    }
    cnt[iSA[n - h]]++;
    b2h[iSA[n - h]] = 1;
    for (int i = 0; i < n; i = nxt[i]) {
      for (int j = i; j < nxt[i]; j++) {
        int s = SA[j] - h;
        if (s < 0) continue;
        int head = iSA[s];
        iSA[s] = head + cnt[head]++;
        b2h[iSA[s]] = 1;
      }
      for (int j = i; j < nxt[i]; j++) {
        int s = SA[j] - h;
        if (s < 0 || !b2h[iSA[s]]) continue;
        for (int k = iSA[s] + 1; !bh[k] && b2h[k]; k++) b2h[k] = 0;
      }
    }
    for (int i = 0; i < n; i++) {
      SA[iSA[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for (int i = 0; i < n; i++) iSA[SA[i]] = i;
}
void buildLCP(int n) {
  for (int i = 0, k = 0; i < n; i++) {
    if (iSA[i] == n - 1) {
      k = 0;
      lcp[n - 1] = 0;
      continue;
    }
    int j = SA[iSA[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
    lcp[iSA[i]] = k;
    if (k) k--;
  }
}
void buildLCPSparse(int n) {
  for (int i = 0; i < n; i++) lcpSparse[0][i] = lcp[i];
  for (int i = 1; i < LOGN; i++) {
    for (int j = 0; j < n; j++) {
      lcpSparse[i][j] = min(lcpSparse[i - 1][j], lcpSparse[i - 1][min(n - 1, j + (1 << (i - 1)))]);
    }
  }
}
pair<int, int> minLCPRange(int n, int from, int minLCP) {
  int r = from;
  for (int i = LOGN - 1; i >= 0; i--) {
    int jump = 1 << i;
    if (r + jump < n and lcpSparse[i][r] >= minLCP) r += jump;
  }
  int l = from;
  for (int i = LOGN - 1; i >= 0; i--) {
    int jump = 1 << i;
    if (l - jump >= 0 and lcpSparse[i][l - jump] >= minLCP) l -= jump;
  }
  return make_pair(l, r);
}
```

6.8 Z Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
char s[N];
int t, n, z[N];
int main() {
  scanf("%s", s);
  n = strlen(s), z[0] = n;
  int L = 0, R = 0;
  for (int i = 1; i < n; ++i) {
    if (i > R) {
      L = R = i;
      while (R < n && s[R - L] == s[R]) ++R;
      z[i] = R - L; --R;
    } else {
      int k = i - L;
      if (z[k] < R - i + 1) z[i] = z[k];
      else {
        L = i;
        while (R < n && s[R - L] == s[R]) ++R;
        z[i] = R - L; --R;
      }
    }
  }
  for (int i = 0; i < n; ++i) {
    printf("%d --> %d\n", i, z[i]);
  }
  return 0;
}
```

7 System

7.1 check

```bash
g++ a.cpp -o a.out && g++ ac.cpp -o ac.out && g++ gen.cpp -o gen && for ((i=0;i<1000;i++)) \
do
echo $i
./gen > in
./a.out < inp > out
./ac.out < inp > out1
diff out1 out2
if [ $? -ne 0 ]
then
echo "-----Input----"
cat in
echo "------Output----"
cat out
echo "-----Accepted----"
cat out1
break
fi
done
```

7.2 compile

```bash
alias rn='g++ -Wall -Wextra -pedantic -std=c++11
    -O2 -Wshadow -Wformat=2 -Wfloat-equal
    -Wconversion -Wlogical-op -Wshift-overflow=2
    -Wduplicated-cond -Wcast-qual -Wcast-align
    -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
    -D_FORTIFY_SOURCE=2 -fsanitize=address
    -fsanitize=undefined -fno-sanitize-recover
    -fstack-protector'
```

7.3 sublimeBuild

```
{
    "shell_cmd": "g++ \"${file}\" -std=c++14 -o
    \"${file_path}/${file_base_name}\" &&
    \"${file_path}/${file_base_name}\" < in > out",
    "working_dir": "${file_path}",
}
```

8 Notes

8.1 Geometry

8.1.1 Triangles

Circumradius: $R = \dfrac{abc}{4A}$, Inradius: $r = \dfrac{A}{s}$
Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$
Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1 - \left(\dfrac{a}{b+c}\right)^2\right]}$

Law of tangents: $\dfrac{a+b}{a-b} = \dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

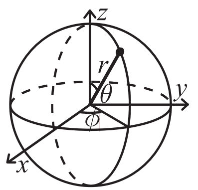
8.1.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin\theta = F\tan\theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

8.1.3 Spherical coordinates



$$\begin{aligned} x &= r\sin\theta\cos\phi & r &= \sqrt{x^2+y^2+z^2} \\ y &= r\sin\theta\sin\phi & \theta &= \mathrm{acos}(z/\sqrt{x^2+y^2+z^2}) \\ z &= r\cos\theta & \phi &= \mathrm{atan2}(y,x) \end{aligned}$$

8.2 Sums

$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$
$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$
$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$
$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}$
$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$

8.3 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \le 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \le x \le 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

$$(x+a)^{-n} = \sum_{k=0}^{\infty}(-1)^k\binom{n+k-1}{k}x^k a^{-n-k}$$

8.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k\cdot(m^2 - n^2), \; b = k\cdot(2mn), \; c = k\cdot(m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

8.5 Number Theory

8.5.1 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.
Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^{\times}$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

8.5.2 Estimates

$\sum_{d|n} d = O(n \log\log n)$.
The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

8.5.3 Perfect numbers

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

8.5.4 Carmichael numbers

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

8.5.5 Mobius function

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

8.5.6 Legendre symbol

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

8.5.7 Jacobi symbol

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

8.5.8 Primitive roots

If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod{m}$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \mathrm{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.
If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod{p}$, $g^u \equiv x \pmod{p}$. $x^n \equiv a \pmod{p}$ iff $g^{nu} \equiv g^i \pmod{p}$ iff $nu \equiv i \pmod{p}$.)

8.5.9 Discrete logarithm problem

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

8.5.10 Pythagorean triples

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

8.5.11 Postage stamps/McNuggets problem

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

8.5.12 Fermat's two-squares theorem

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

8.6 Permutations

8.6.1 Factorial

n	1 2 3 4	5	6	7	8	9	10	
$n!$	1 2 6 24	120	720	5040	40320	362880	3628800	
n	11	12	13	14	15	16	17	
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14	
n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

8.6.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!}=\exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

8.6.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n)=(n-1)(D(n-1)+D(n-2))=nD(n-1)+(-1)^n=\left\lfloor\frac{n!}{e}\right\rceil$$

8.6.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g $(g.x=x)$.
If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G=\mathbb{Z}_n$ to get

$$g(n)=\frac{1}{n}\sum_{k=0}^{n-1}f(\gcd(n,k))=\frac{1}{n}\sum_{k|n}f(k)\phi(n/k)$$

8.7 Partitions and subsets

8.7.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0)=1,\ p(n)=\sum_{k\in\mathbb{Z}\setminus\{0\}}(-1)^{k+1}p(n-k(3k-1)/2)$$

$$p(n)\sim 0.145/n\cdot\exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

8.8 General purpose numbers

8.8.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k)=c(n-1,k-1)+(n-1)c(n-1,k),\ c(0,0)=1$$
$$\textstyle\sum_{k=0}^{n}c(n,k)x^k=x(x+1)\dots(x+n-1)$$

$c(8,k)=8,0,5040,13068,13132,6769,1960,322,28,1$
$c(n,2)=0,0,1,3,11,50,274,1764,13068,109584,\dots$

8.8.2 Eulerian numbers

Number of permutations $\pi\in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j)>\pi(j+1)$, $k+1$ j:s s.t. $\pi(j)\geq j$, k j:s s.t. $\pi(j)>j$.

$$E(n,k)=(n-k)E(n-1,k-1)+(k+1)E(n-1,k)$$

$$E(n,0)=E(n,n-1)=1$$

$$E(n,k)=\sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

8.8.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

$$S(n,1)=S(n,n)=1$$

$$S(n,k)=\frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

8.8.4 Bell numbers

Total number of partitions of n distinct elements. $B(n)=1,1,2,5,15,52,203,877,4140,21147,\dots$. For p prime,

$$B(p^m+n)\equiv mB(n)+B(n+1)\pmod p$$

8.8.5 Bernoulli numbers

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

8.8.6 Catalan numbers

$$C_n=\frac{1}{n+1}\binom{2n}{n}=\binom{2n}{n}-\binom{2n}{n+1}=\frac{(2n)!}{(n+1)!n!}$$

$$C_0=1,\ C_{n+1}=\frac{2(2n+1)}{n+2}C_n,\ C_{n+1}=\sum C_iC_{n-i}$$

$C_n=1,1,2,5,14,42,132,429,1430,4862,16796,58786,\dots$

- sub-diagonal monotone paths in an $n\times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

8.9 Inequalities

8.9.1 Titu's Lemma

For positive reals $a_1,a_2,\dots,a_n$ and $b_1,b_2,\dots,b_n$,

$$\frac{{a_1}^2}{b_1}+\frac{{a_2}^2}{b_2}+\dots+\frac{{a_n}^2}{b_n}\geq\frac{a_1+a_2+\dots+{a_n}^2}{b_1+b_2+\dots+b_n}$$

Equality holds if and only if $a_i=kb_i$ for a non-zero real constant k.

8.10 Games

8.10.1 Grundy numbers

For a two-player, normal-play (last to move wins) game on a graph (V,E): $G(x)=\text{mex}(\{G(y):(x,y)\in E\})$, where $\text{mex}(S)=\min\{n\geq 0:n\not\in S\}$. x is losing iff $G(x)=0$.

8.10.2 Sums of games

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

8.10.3 Misère Nim

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