Contents

1 Codes

1.1 2 SAT

```
namespace sat{
  const int MAX = 200010; /// number of variables * 2
  int n, m; bool vis[MAX]; int comp[MAX];
  vector <int> ed[MAX], rev[MAX], order;
  inline int inv(int x) { return ((x) <= n ? (x + n) : (x - n)); }
  /// Call init once
  void init(int vars) {
      n = vars, m = vars << 1;
      for (int i = 1; i <= m; i++) {
          ed[i].clear(); rev[i].clear(); }}
  /// Adding implication, if a then b ( a --> b )
  inline void add(int a, int b) {
      ed[a].push_back(b);
      rev[b].push_back(a); }
  inline void OR(int a, int b) { add(inv(a), b); add(inv(b), a); }
  inline void AND(int a, int b) { add(a, b); add(b, a); }
  void XOR(int a,int b) { add(inv(b), a); add(a, inv(b)); add(inv(a), b); add(b, inv(a)); }
  inline void XNOR(int a, int b) { add(a,b); add(b,a); add(inv(a), inv(b)); add(inv(b), inv(a)); }
  /// (x <= n) means forcing variable x to be true
  /// (x = n + y) means forcing variable y to be false
  inline void force_true(int x) { add(inv(x), x); }

  void dfs(int s) { vis[s] = true;
    for(int x : ed[s]) {
        if(vis[x]) continue; dfs(x); }
    order.push_back(s); }

  void mark(int s, int c) {
    comp[s] = c; vis[s] = true;
    for(int x : rev[s]) {
      if(vis[x]) continue; mark(x, c); }}

  bool satisfy(vector <int> &res){
    CLR(vis); order.clear();
    for(int i=1;i<=m;i++) {
      if(vis[i]) continue; dfs(i);
    } CLR(vis); int it = 0;
    for(int i=m-1;i>=0;i--) {
      int s = order[i];
      if(vis[s]) continue;
      mark(s, ++it); }
    res.clear();
    for(int i=1;i<=n;i++) {
      if(comp[i] == comp[inv(i)]) return false;
      if(comp[i] > comp[inv(i)])
        res.push_back(i);
    } return true; }}
```

1.2 2D Pattern Matcher

```
int matching(){
  int i, j, ch, cnt, k;
  for(i = 0; i < n; i++){
    for(j =0; j < m; j++){
      resMatrix[i][j] = 1; } }
  for(ch = 'a'; ch <= 'b'; ch++){
    patternVec.clear(); strVec.clear(); mulVec.clear(); cnt = 0;
    for(i = 0; i < n; i++){ for(j =0; j < m; j++){
        if(i <= r - 1 && j <= c - 1){
          if(pattern[i][j] == ch){
              patternVec.push_back(1); cnt++; } else{patternVec.push_back(0);}
        } else{patternVec.push_back(0);}
        if(str[i][j] == ch){strVec.push_back(1);} else{strVec.push_back(0);}}}
    reverse(patternVec.begin(), patternVec.end()); polMul(strVec, patternVec, mulVec);
    for(i = 0; i <= n - r; i++){
      for(j = 0; j <= m - c; j++){
        if(i*m + j + m*n - 1 < (int)mulVec.size()){
          resMatrix[i][j] &= (mulVec[i*m + j + m*n - 1] == cnt); }
        else{ resMatrix[i][j] &= 0; }}}}
  for(i = 0, cnt = 0; i <= n - r; i++){
    for(j = 0; j <= m - c; j++){ cnt += resMatrix[i][j];}} return cnt; }
```

1.3 3D Convex Hull + Some 3D Geo Routines

```
const double eps = 1e-12;
int dcmp(double val) {
    if(fabs(val) < eps) return 0;
    if(val < 0) return -1;
    return 1;
}
struct Point3 { double x, y, z;
  Point3 (double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}
  bool operator < (const Point3 &u) const
      { return dcmp(x - u.x) < 0 || dcmp(x - u.x) == 0 && dcmp(y - u.y) < 0; }
  bool operator > (const Point3 &u) const { return u < (*this); }
  bool operator == (const Point3 &u)
      const { return !(u < (*this) || (*this) < u); }
  bool operator != (const Point3 &u) const { return !((*this) == u); }
  Point3 operator + (const Point3 &u)
      const { return Point3(x + u.x, y + u.y, z + u.z); }
  Point3 operator - (const Point3 &u)
      const { return Point3(x - u.x, y - u.y, z - u.z); }
  Point3 operator * (const double u)
      const { return Point3(x * u, y * u, z * u); }
  Point3 operator / (const double u)
      const { return Point3(x / u, y / u, z / u); }
};
typedef Point3 Vector3;
namespace Vectorial {
  double getDot(Vector3 a, Vector3 b) {
    return (a.x * b.x + a.y * b.y + a.z * b.z);
  }
  Vector3 getCross(Vector3 a, Vector3 b) {
    return Vector3 (a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - b.x * a.y);
  }
  double getLength(Vector3 a) {
    return sqrt( getDot(a, a) );
  }
  //Next 3 functions are not needed in CH3D
  double getAngle(Vector3 a, Vector3 b){
    return acos(getDot(a, b) / getLength(a) / getLength(b));
  }
  double getDistanceToPlane(Point3 p, Point3 p0, Vector3 &v){
    return fabs(getDot(p - p0, v));
  }
```

```
  Point3 getPlaneProjection(Point3 p,
   ↪  Point3 p0, Vector3 v) {
    return p - v * getDot(p - p0, v);
  }
};
//Linear not needed in CH3D
namespace Linear {
  using namespace Vectorial;
  double getDistanceToLine(Point3 p,
   ↪  Point3 a, Point3 b) {
    Vector3 v1 = b - a, v2 = p - a;
    return getLength(getCross(v1, v2)) /
     ↪  getLength(v1);
  }
  double getDistanceToSegment(Point3 p,
   ↪  Point3 a, Point3 b){
    if(a == b) return getLength(p - a);
    Vector3 v1 = b - a, v2 = p - a, v3 =
     ↪  p - b;
    if(dcmp(getDot(v1, v3)) > 0) return
     ↪  getLength(v3);
    else return getLength( getCross(v1,
     ↪  v2) ) / getLength(v1);
  }
}
namespace Triangular {
  using namespace Vectorial;
  double getArea(Point3 a, Point3 b,
   ↪  Point3 c) {
    return getLength(getCross(b - a, c -
     ↪  a));
  }
  //The Next 2 functions are not needed
   ↪  in CH3D
  bool onTriangle(Point3 p, Point3 a,
   ↪  Point3 b, Point3 c) {
    double area1 = getArea(p, a, b),
     ↪  area2 = getArea(p, b, c);
    double area3 = getArea(p, c, a);
    return dcmp(area1 + area2 + area3 -
     ↪  getArea(a, b, c)) == 0;
  }
  bool haveIntersectionTriSeg(Point3 p0,
      Point3 p1, Point3 p2, Point3 a,
   ↪  Point3 b, Point3& p){
    Vector3 v = getCross(p1 - p0, p2 -
     ↪  p0);
    if(dcmp(getDot(v, b - a)) == 0)
     ↪  return false;
    else {
      double t = getDot(v, p0 - a) /
       ↪  getDot(v, b - a);
      if(dcmp(t) < 0 || dcmp(t - 1) > 0)
       ↪  return false;
      p = a + (b - a) * t;
      return onTriangle(p, p0, p1, p2);
    }
  }
};
namespace Polygonal {
  using namespace Vectorial;
  using namespace Triangular;
  struct Face{
    int v[3];
    Face(int a = 0, int b = 0, int c = 0){
      v[0] = a, v[1] = b, v[2] = c;
    }
    Vector3 normal (Point3 *p) const {
      return getCross(p[v[1]] - p[v[0]],
       ↪  p[v[2]] - p[v[0]]);
    }
    bool cansee(Point3 *p, int i) const {
      return getDot(p[i] - p[v[0]],
       ↪  normal(p)) > 0 ? 1 : 0;
    }
  };
  double getVolume (Point3 a, Point3 b,
   ↪  Point3 c, Point3 d) {
    return getDot(d - a, getCross(b - a,
     ↪  c - a)) / 6;
  }
  vector<Face> CH3D (Point3 *p, int n) {
    bool vis[MAX][MAX];
    vector<Face> cur;
    memset(vis, 0, sizeof (vis));
    cur.push_back({0, 1, 2});
    cur.push_back({2, 1, 0});
    for(int i = 3; i < n; i++) {
      vector<Face> net;
      for(int j = 0; j < cur.size(); j++)
       ↪  {
        Face& fa = cur[j];
        bool res = fa.cansee(p, i);
        if(!res) net.push_back(fa);
        for(int k = 0; k < 3; k++){
          vis[fa.v[k]][fa.v[(k + 1) % 3]]
           ↪  = res;
        }
      }
      for(int j = 0; j < cur.size(); j++)
       ↪  {
        for(int k = 0; k < 3; k++){
          int a = cur[j].v[k], b =
           ↪  cur[j].v[(k + 1) % 3];
          if(vis[a][b] != vis[b][a] &&
           ↪  vis[a][b]) {
            net.push_back({a, b, i});
          }
        }
      }
      cur = net;
    }
    return cur;
  }
  double getCH3Dsurface(Point3 *p,
   ↪  std::vector<Face> CH3D) {
    double ret = 0;
    for(int i = 0; i < CH3D.size(); i++){
      ret += getArea(p[CH3D[i].v[0]],
          p[CH3D[i].v[1]],
       ↪  p[CH3D[i].v[2]]) * .5;
    }
    return ret;
  }
  double getCH3Dvolume(Point3 *p,
   ↪  std::vector<Face> CH3D) {
    Point3 O = p[CH3D[0].v[0]];
    double ret = 0;
    for(int i = 1; i < CH3D.size(); i++){
      ret += abs(getVolume(O,
          p[CH3D[i].v[0]],
       ↪  p[CH3D[i].v[1]],
       ↪  p[CH3D[i].v[2]]) );
    }
    return ret;
  }
};
using namespace Polygonal;
Point3 P[1010];
int main() {
  int t, n; scanf("%d", &t);
  while(t--) {
    scanf("%d", &n);
    for(int i = 0; i < n; i++) scanf("%lf
     ↪  %lf %lf",&P[i].x, &P[i].y,
     ↪  &P[i].z);
    std::vector<Face> CH = CH3D(P, n);
    printf("%.9lf %.9lf\n",
        getCH3Dsurface(P, CH),
     ↪  getCH3Dvolume(P, CH));
  } return 0;
}
```

1.4 Aho Corasick

```
struct AC {
int N, P; const int A = 26;
vector <int> link, out_link, cnt, ans;
vector < vector <int> > out, ed, next;
AC(): N(0), P(0) { node(); }
```

```
int node() {
  next.emplace_back(A, 0); link.emplace_back(0);
  out.emplace_back(0); out_link.emplace_back(0);
  cnt.emplace_back(0); ed.emplace_back(0);
  return N++; }
void clear() {
  next.clear(), link.clear(), out.clear(), out_link.clear(), ed.clear();
  cnt.clear(), ans.clear(); N = P = 0; node();}
inline int get(char c) { return c - 'a'; }
void insert(const string &T) {
  int u = 0;
  for (char c : T) {
    if (!next[u][get(c)]) next[u][get(c)] = node();
    u = next[u][get(c)];
  } out[u].push_back(P); ans.push_back(0); P++; }
void build() {
  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];
        ed[link[v]].push_back(v); q.push(v);
      }}}}
void dfs(int s) {
  for(int x : ed[s]) dfs(x), cnt[s] += cnt[x];
  for(int e : out[s]) ans[e] = cnt[s];}
void traverse(const string &S) {
  int u = 0; for (char c : S) {
    u = next[u][get(c)];
    cnt[u]++; } dfs(0); }
};
// AC aho; aho.insert(pat); aho.build(); aho.traverse(text); aho.clear();
```

1.5 Area of Union of N Circles

```
#include <bits/stdc++.h>
using namespace std;
#define N 105 //circles
struct Point{
  double x,y;
  Point(double a = 0.0, double b = 0.0) : x(a) , y(b) {}
  Point operator+(const Point&a) const { return Point(x + a.x,y + a.y); }
  Point operator-(const Point&a) const { return Point(x - a.x,y - a.y); }
  Point operator*(const double&a) const { return Point(x * a,y * a); }
  Point operator/(const double&a) const { return Point(x / a,y / a); }
  double operator*(const Point&a) const { return x * a.y - y * a.x; }
  double operator/(const Point&a) const {
    return hypot(a.x-x, a.y-y);} //distance
} po[N];
double r[N];
const double pi = acos(-1.00), eps = 1e-7;
inline int sgn(double x) {return fabs(x) < eps ? 0 :(x > 0.0 ? 1 : -1);}
pair<double,bool>ARG[2*N];
double cir_uni(int n){
  double sum = 0.0, sum1 = 0.0, d, p1, p2, p3;
  for(int i = 0; i < n; i++){
    bool f = 1;
    for(int j = 0; f && j < n; j++){
      if( i != j && sgn(r[j] - r[i] - po[i]/po[j]) != -1) f = 0;
    }
    if(!f)swap(r[i],r[--n]), swap(po[i--],po[n]);
  }
  for(int i = 0;i < n;i++){
    int k = 0, cnt = 0;
    for(int j = 0; j < n; j++){
      if(i != j && sgn((d = po[i]/po[j]) - r[i] - r[j]) <= 0) {
        p3 = acos((r[i]*r[i]+d*d-r[j]*r[j])/(2.0 * r[i]*d));
        p2 = atan2(po[j].y-po[i].y,po[j].x-po[i].x);
        p1 = p2 - p3; p2 = p2 + p3;
        if(sgn( p1 + pi) == -1)p1 += 2*pi,cnt++;
        if(sgn(p2 - pi) == 1)p2 -= 2*pi, cnt++;
        ARG[k++] = {p1,0};
        ARG[k++] = {p2,1};
      }
    }
    if(k){
      sort(ARG, ARG+k);
      p1 = ARG[k-1].first - 2*pi;
      p3 = r[i]*r[i];
      for(int j = 0; j < k; j++){
        p2 = ARG[j].first;
        if(cnt == 0){
          sum += (p2-p1-sin(p2-p1))*p3;
          sum1 += (po[i] + Point(cos(p1), sin(p1))*r[i])*(po[i] + Point(cos(p2), sin(p2))*r[i]);
        }
        p1 = p2;
        ARG[j].second ? cnt--: cnt++; }
    }
    else sum += 2*pi*r[i]*r[i]; }
  return (sum+fabs(sum1))*0.5;
}
```

1.6 BIT Range

```
// Add v to A[a...b]
update(a, b, v): update(B1, a, v)
  update(B1, b + 1, -v) update(B2, a, v * (a-1)) update(B2, b + 1, -v * b)
// Return sum A[1...b]
query(b): return query(B1, b) * b - query(B2, b)
```

1.7 Block Cut Tree

```
// clear ed[],tree[] every test case
// tot -> total number of components
// nn -> number of nodes in the Block Cut Tree (<= n + n)
// bcc[i] contains the nodes of the i'th component
// tree[] is the edge list of BCT
// c cut vertices, Nodes([1,tot]) -> the biconnected components
// Nodes([tot+1,tot+c]) -> the cut vertices, nn = tot + c
// If x is NOT a cut vertex, compNum[x] is the index of the bcc[] x is present in
// If x is a cut vertex, compNum[x]( > tot ) is the index of x in BCT
const int MAX = ?; vector <int> ed[MAX];
bool cut[MAX]; int tot, Time, low[MAX], st[MAX];
vector <int> bcc[MAX]; stack <int> S;
```

```
void popBCC(int s,int x) {
  cut[s] = 1; bcc[++tot].pb(s);
  while(bcc[tot].back() ^ x) {
    bcc[tot].pb(S.top()); S.pop(); }}
void dfs(int s, int p = -1) {
  S.push(s); int ch = 0; st[s] = low[s] = ++Time;
  for(int x : ed[s]) { if(!st[x]) {
      ch++; dfs(x,s);
      low[s] = min(low[s],low[x]);
      if(p != -1 && low[x] >= st[s]) popBCC(s,x);
      else if(p == -1) if(ch > 1) popBCC(s,x); }
    else if(p != x) low[s] = min(low[s],st[x]); }
  if(p == -1 && ch > 1) cut[s] = 1; }
void processBCC(int n) {
  for(int i=1;i<=n;i++) bcc[i].clear();
  CLR(st); CLR(cut); Time = tot = 0;
  for(int i=1; i<=n; i++) {
    if(!st[i]) { dfs(i,-1);
      if(!S.empty()) ++tot;
      while(!S.empty()) {
        bcc[tot].push_back(S.top()); S.pop(); }}}}

vector <int> tree[MAX + MAX];
int compNum[MAX]; int nn;

void buildTree(int n) { processBCC(n); nn = tot;
    for(int i=1;i<=n;i++) if(cut[i]) compNum[i] = ++nn;
    for(int i=1;i<=tot;i++) {
      for(int v : bcc[i]) { if(cut[v]) {
          tree[i].pb(compNum[v]);
          tree[compNum[v]].pb(i);
        } else compNum[v] = i;}}}
```

1.8 Bridge Tree

```
// Clear ed , isBridge , brTree per test case
const int MAXN = ?; const int MAXE = ?;
struct edges { int u, v; } ara[MAXE];
vector <int> ed[MAXN], isBridge[MAXN], brTree[MAXN];
bool vis[MAXN]; int st[MAXN], low[MAXN], Time = 0;
int cnum, comp[MAXN];
void findBridge(int s, int par) {
  int i, x, child = 0, j;
  vis[s] = 1; Time++;
  st[s] = low[s] = Time;
  for(i=0; i<ed[s].size(); i++) {
    x = ed[s][i];
    if(!vis[x]) { child++;
      findBridge(x,s); low[s] = min(low[s],low[x]);
      if(low[x] > st[s]) { isBridge[s][i] = 1;
        j = lower_bound(ed[x].begin(),ed[x].end(),s)-ed[x].begin();
        isBridge[x][j] = 1; }}
    else if(par!=x) low[s] = min(low[s],st[x]); }}
void dfs(int s) {
    int i, x; vis[s] = 1; comp[s] = cnum;
    for(i=0; i<ed[s].size(); i++) {
      if(!isBridge[s][i]) { x = ed[s][i];
        if(!vis[x]) dfs(x); }}}
void processBridge(int n,int m) {
  CLR(vis); Time = 0;
  for(int i=1; i<=n; i++) if(!vis[i]) findBridge(i,-1);
  cnum = 0; CLR(vis);
  for(int i=1; i<=n; i++) {
    if(!vis[i]) { cnum++; dfs(i); }}
  n = cnum;
  for(int i=1; i<=m; i++) {
    if(comp[ara[i].u] != comp[ara[i].v]) {
      brTree[comp[ara[i].u]].pb(comp[ara[i].v]);
      brTree[comp[ara[i].v]].pb(comp[ara[i].u]);
    }}}
int main() { int n,m,u,v;
  scanf("%d %d",&n,&m);
  for(int i=1; i<=m; i++) {
    sii(u,v); ed[u].pb(v); ed[v].pb(u);
    isBridge[u].pb(0); isBridge[v].pb(0);
    ara[i].u = u; ara[i].v = v; }
  for(int i=1; i<=n; i++) sort(all(ed[i]));
  processBridge(n,m); return 0; }
```

1.9 Centroid Decomposition

```
int nn; vector <int> ed[MAX]; bool isCentroid[MAX];
int sub[MAX], cpar[MAX], clevel[MAX]; int dis[20][MAX];
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); } }
// decompose(1, -1, 0)
```

1.10 Closest Pair of Points

```
//Returns square distance
#define ll long long
#define x first
#define y second
typedef pair<int,int> point;
ll sq(double n){ return n * n;}
ll dist(point a, point b){
  return sq(a.x - b.x)  + sq(a.y - b.y);
}
vector<point> p;
ll solve(int l, int r) {
  if(r - l <= 3) {
    ll ret = 1e18;
    for(int i = l; i <= r; i++)
      for(int j = i + 1; j <= r; j++)
        ret = min(ret, dist(p[i], p[j]));
    return ret;
  }
  int mid = l + r >> 1;
  ll d = min(solve(l, mid), solve(mid+1, r));
  vector<point> t;
```

```cpp
  for(int i = l; i <= r; i++)
  if(sq(p[mid].x - p[i].x) <= d)
    t.push_back({p[i].y, p[i].x});
  sort(t.begin(), t.end());
  for(int i = 0; i < t.size(); i++) {
    for(int j = i+1; j < t.size() && j <= i + 15; j++)
      d = min(d, dist(t[i], t[j]));
  } return d;
}
ll closestPair() {
  sort(p.begin(), p.end());
  return solve(0, p.size()-1);
}
int main() {
  int n;
  cin >> n;
  p.resize(n);
  for(int i = 0;i < n; i++){
    cin >> p[i].first >> p[i].second;
  }
  double ans = closestPair();
  ans = sqrt(ans);
  printf("%.6lf\n", ans);
  return 0;
}
```

1.11 Convex Hull

```cpp
typedef long long ll;
struct point{
 int x, y;
 bool operator < (const point &p) const {
  return x == p.x ? y < p.y : x < p.x;}};
ll cross (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) {
 int sz = 0; vector<point> hull(n + n);
 sort(p.begin(), p.end());
  for(int i = 0; i < n; ++i) {
  while (sz > 1 and cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz;
  hull[sz++] = p[i]; }
 for(int i = n - 2, j = sz + 1; i >= 0; --i) {
  while (sz >= j and cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz;
  hull[sz++] = p[i]; }
 hull.resize(sz - 1); return hull; }
```

1.12 DCHT

```cpp
const long long LL_INF = (long long) 2e18 + 5;
struct point {
  long long x, y; point() : x(0), y(0) {}
  point(long long _x, long long _y) : x(_x), y(_y) {}
};
// dp_hull enables you to do the following two operations in amortized O(log n) time:
// 1. Insert a pair (a_i, b_i) into the structure
// 2. For any value of x, query the maximum value of a_i * x + b_i
// All values a_i, b_i, and x can be positive or negative.
struct dp_hull {
  struct segment {
    point p;
    mutable point next_p;
    segment(point _p = {0, 0}, point _next_p = {0, 0}) : p(_p), next_p(_next_p) {}
    bool operator<(const segment &other) const {
      // Sentinel value indicating we should binary search the set for a single x-value.
      if (p.y == LL_INF)
        return p.x * (other.next_p.x - other.p.x) <= other.p.y - other.next_p.y;
      return make_pair(p.x, p.y) < make_pair(other.p.x, other.p.y);
    }
  };
  set&lt;segment&gt; segments;
  int size() const { return segments.size(); }
  set&lt;segment&gt;::iterator prev(set&lt;segment&gt;::iterator it) const {
    return it == segments.begin() ? it : --it; }
  set&lt;segment&gt;::iterator next(set&lt;segment&gt;::iterator it) const {
    return it == segments.end() ? it : ++it; }
  static long long floor_div(long long a, long long b) {
    return a / b - ((a ^ b) < 0 && a % b != 0); }
static bool bad_middle(const point &a, const point &b, const point &c) {
  // This checks whether the x-value where b beats a comes after the x-value where c beats b. It's fine to round
  // down here if we will only query integer x-values. (Note: plain C++ division rounds toward zero)
  return floor_div(a.y - b.y, b.x - a.x) >= floor_div(b.y - c.y, c.x - b.x);
}
bool bad(set&lt;segment&gt;::iterator it) const {
  return it != segments.begin() && next(it) != segments.end() && bad_middle(prev(it)->p, it->p, next(it)->p);
}
void insert(const point &p) {
  set&lt;segment&gt;::iterator next_it = segments.lower_bound(segment(p));
  if (next_it != segments.end() && p.x == next_it->p.x) return;
  if (next_it != segments.begin()) {
    set&lt;segment&gt;::iterator prev_it = prev(next_it);
    if (p.x == prev_it->p.x) segments.erase(prev_it);
    else if (next_it != segments.end() && bad_middle(prev_it->p, p, next_it->p)) return;
  }
  // Note we need the segment(p, p) here for the single x-value binary search.
  set&lt;segment&gt;::iterator it = segments.insert(next_it, segment(p, p));
  while (bad(prev(it))) segments.erase(prev(it));
  while (bad(next(it))) segments.erase(next(it));
  if (it != segments.begin()) prev(it)->next_p = it->p;
  if (next(it) != segments.end()) it->next_p = next(it)->p;
}
void insert(long long a, long long b) { insert(point(a, b)); }
```

```
  // Queries the maximum value of ax + b.
  long long query(long long x) const {
    assert(size() > 0);
    set&lt;segment&gt;::iterator it = segments.upper_bound(segment(point(x, LL_INF)));
    return it->p.x * x + it->p.y;
  }
};
```

1.13 DSU On Tree

```
vector <int> G[MAX];
int n, sub[MAX], color[MAX], freq[MAX];
void calcSubSize(int s,int p) {
  sub[s] = 1; for(int x : G[s]) {
    if(x==p) continue; calcSubSize(x,s);
    sub[s] += sub[x]; }}
void add(int s,int p,int v,int bigchild = -1) {
  freq[color[s]] += v; for(int x : G[s]) {
    if(x==p || x==bigchild) continue;
      add(x,s,v);
  }}
void dfs(int s,int p,bool keep) {
  int bigChild = -1;
  for(int x : G[s]) { if(x==p) continue;
    if(bigChild==-1 || sub[bigChild] < sub[x] ) bigChild = x; }
  for(int x : G[s]) {
    if(x==p || x==bigChild) continue;
      dfs(x,s,0); }
  if(bigChild!=-1) dfs(bigChild,s,1);
  add(s,p,1,bigChild);
  /// freq[c] now contains only the info of the subtree of node c
  if(keep==0) add(s,p,-1); }
// input; calcSubSize(root,-1); dfs(root,-1,0);
```

1.14 Dinic

```
namespace dinic {
using T = int; const T INF = 2e9; const int MAXN = 5010;
int n, src, snk, work[MAXN]; T dist[MAXN];
struct Edge{ int to, rev_pos; T c, f; };
vector <Edge> ed[MAXN];
void init(int _n, int _src, int _snk) {
  n = _n, src = _src, snk = _snk;
  for(int i=0;i<=n;i++) ed[i].clear();}
inline void addEdge(int u, int v, T c, T rc = 0) {
  Edge a = {v, ed[v].size(), c, 0};
  Edge b = {u, ed[u].size(), rc, 0};
  ed[u].push_back(a); ed[v].push_back(b);
}
bool dinic_bfs(){
  SET(dist); dist[src] = 0; queue <int> q; q.push(src);
  while(!q.empty()){ int u = q.front(); q.pop();
    for(Edge &e : ed[u]){
      if(dist[e.to] == -1 and e.f < e.c){
        dist[e.to] = dist[u] + 1;
          q.push(e.to); } } }
  return (dist[snk]>=0); }
T dinic_dfs(int u, T fl){
  if (u == snk) return fl;
  for (; work[u] < ed[u].size(); work[u]++){
    Edge &e = ed[u][work[u]];
    if (e.c <= e.f) continue; int v = e.to;
    if (dist[v] == dist[u] + 1){
      T df = dinic_dfs(v, min(fl, e.c - e.f));
      if (df > 0){ e.f += df;
        ed[v][e.rev_pos].f -= df;
        return df; }}} return 0; }
T solve() {
    T ret = 0; while (dinic_bfs()){
      CLR(work);
      while (T delta = dinic_dfs(src, INF)) ret += delta;
    } return ret; } }
// init -> edge input -> solve
```

1.15 DnC Opt

```
// given n objects with weights w1,w2,...,wn, divide them into m groups of consecutive objects, such that the sum of squares of total weights of the groups is minimal
def ComputeDP(i, jleft, jright, kleft, kright):
  // Select the middle point
  jmid = (jleft + jright) / 2
  // Compute the value of dp[i][jmid] by definition of DP
  dp[i][jmid] = +INFINITY
  bestk = -1
  for k in range(kleft, jmid):
    if dp[i - 1][k] + C[k + 1][jmid] < best:
      dp[i][jmid] = dp[i - 1][k] + C[k + 1][jmid]
      bestk = k
  // Divide and conquer
  if jleft < jmid - 1:
    ComputeDP(i, jleft, jmid - 1, kleft, bestk)
  if jleft + 1 < jright:
    ComputeDP(i, jmid + 1, jright, bestk, kright)
def ComputeFullDP:
  Initialize dp for i = 0 somehow
  for i in range(1, m):
    ComputeDP(i, 0, n, 0, n)
```

1.16 Dominator Tree

```
// init() at the start of testcase, 1 based
// tree->dom tree, g-> actual graph
// If a problem asks for edge disjoint paths, for every edge, take a new node w and  turn the  edge (u --> v) to (u --> w --> v) and find node disjoint path now.
const int MAX = 200010;
vector<int> g[MAX+5],tree[MAX+5],rg[MAX+5],bucket[MAX+5];
int sdom[MAX+5],par[MAX+5],dom[MAX+5],dsu[MAX+5],label[MAX+5];
int arr[MAX+5],rev[MAX+5], Time ,n, source;
void init(int _n, int _source){
  Time = 0; n = _n; source = _source;
  for(int i = 1; i<=n; i++) {
    g[i].clear(), rg[i].clear(), tree[i].clear(), bucket[i].clear();
    arr[i] = sdom[i] = par[i] = dom[i] = dsu[i] = label[i] = rev[i] = 0; }}
void dfs(int u) {
  Time++; arr[u] = Time;
  rev[Time] = u; label[Time] = Time;
  sdom[Time] = Time; dsu[Time] = Time;
    int i,w;
  for(i=0; i<g[u].size(); i++) { w = g[u][i];
    if(!arr[w]) { dfs(w);
      par[arr[w]] = arr[u]; }
    rg[arr[w]].push_back(arr[u]); }}
inline int Find(int u,int x = 0) {
  if(u == dsu[u]) return x ? -1 : u;
  int v = Find(dsu[u],x+1);
  if(v<0) return u;
```

```cpp
  if(sdom[label[dsu[u]]] < sdom[label[u]])
    label[u] = label[dsu[u]];
  dsu[u] = v;
  return x ? v : label[u];
}
inline void Union(int u,int v){ dsu[v] = u;}
void build(){
  dfs(source);
  for(int i=n; i>=1; i--) {
    for(int j=0; j<rg[i].size(); j++)
      sdom[i] = min(sdom[i],sdom[Find(rg[i][j])]);
    if(i>1)bucket[sdom[i]].push_back(i);
    for(int j=0; j<bucket[i].size(); j++) {
      int w = bucket[i][j],v = Find(w);
      if(sdom[v]==sdom[w]) dom[w]=sdom[w];
      else dom[w] = v;
    } if(i>1) Union(par[i],i);
  }
  for(int i=2; i<=n; i++) {
    if(dom[i]!=sdom[i])dom[i]=dom[dom[i]];
    /// comment the following line out if you don't want bidirectional edges in dominator tree
    tree[rev[i]].push_back(rev[dom[i]]);
    tree[rev[dom[i]]].push_back(rev[i]);
  }
}
```

1.17 DynamicDiameter

```cpp
// Diameter with online positive edge weight updates, O(n lg n), f[x] = distance from root to x, x <= y <= z, max{f[x] - 2f[y] + f[z]}, a --> f[x], b --> -2f[y], c --> f[x] - 2f[y], d --> -2f[y] + f[z], e --> f[x] - 2f[y] + f[z]
struct node {
  ll a = 0, b = 0, c = 0, d = 0, e = 0, lazy = 0;
  node operator + (const node &oth) const { node ret;
    ret.a = max(a, oth.a); ret.b = max(b, oth.b);
    ret.c = max(max(c, oth.c), a + oth.b);
    ret.d = max(max(d, oth.d), b + oth.a);
    ret.e = max(max(e, oth.e), max(c + oth.a, a + oth.d));
    return ret; }};
const int N = 200010; ll mod, W[N]; node t[4 * N]; vector <int> g[N]; int n, q, U[N], V[N], ptr, l[N], r[N], pos[N];
void dfs (int u = 1, int from = 0) {
  l[u] = ++ptr; for (int e : g[u]) {
    int v = U[e] ^ u ^ V[e]; if (v == from) continue; pos[e] = v;
    dfs(v, u); ++ptr; }
  r[u] = ptr; }
void push (int u, int b, int e) {
  ll v = t[u].lazy;
  t[u].a += v, t[u].b -= v + v, t[u].c -= v, t[u].d -= v;
  if (b ^ e) t[u << 1].lazy += v, t[u << 1 | 1].lazy += v;
  t[u].lazy = 0; }
void update (int l, int r, ll v, int u = 1, int b = 1, int e = n + n) {
  if (t[u].lazy) push(u, b, e); if (b > r or e < l) return;
  if (b >= l and e <= r) { t[u].lazy += v; return push(u, b, e); }
  int mid = b + e >> 1;
  update(l, r, v, u << 1, b, mid);
  update(l, r, v, u << 1 | 1, mid + 1, e);
  t[u] = t[u << 1] + t[u << 1 | 1]; }
int main() {
  cin >> n >> q >> mod;
  // scan U[i], V[i], W[i], g[U[i]].emplace_back(i), g[V[i]].emplace_back(i);
  dfs(); for (int i = 1; i < n; ++i)
    update(l[pos[i]], r[pos[i]], W[i]);
  ll last = 0; while (q--) {
    int d; ll e; scanf("%d %lld", &d, &e);
    d = 1 + (d + last) % (n - 1); e = (e + last) % mod;
    update(l[pos[d]], r[pos[d]], e - W[d]); last = t[1].e, W[d] = e;
    printf("%lld\n", last); }}
```

1.18 EGCD

```cpp
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) { x = 0; y = 1; return b; } int x1, y1; int d = gcd(b%a, a, x1, y1); x = y1 - (b / a) * x1; y = x1;
    return d; }
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0); if (c % g) { return false; } x0 *= c / g; y0 *= c / g; if (a < 0) x0 = -x0; if (b < 0) y0 = -y0; return true; }
void shift_solution(int & x, int & y, int a, int b, int cnt) { x += cnt * b; y -= cnt * a; }
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) { int x, y, g;
  if (!find_any_solution(a, b, c, x, y, g)){ return 0; } a /= g; b /= g;
  int sign_a = a > 0 ? +1 : -1; int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx){ shift_solution(x, y, a, b, sign_b); }
  if (x > maxx){ return 0; } int lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx){ shift_solution(x, y, a, b, -sign_b); } int rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny){ shift_solution(x, y, a, b, -sign_a); }
  if (y > maxy){ return 0; } int lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy){ shift_solution(x, y, a, b, sign_a); }
  int rx2 = x; if (lx2 > rx2){ swap(lx2, rx2); }
  int lx = max(lx1, lx2); int rx = min(rx1, rx2); if (lx > rx){ return 0; }
  return (rx - lx) / abs(b) + 1;
}
```

1.19 Euler Trail(Directed)

```cpp
// 1 based, fill edge, call find euler
const int MAX = ?; bool vis[MAX+5];
vector <int> ed[MAX+5], sltn;
int in[MAX+5], out[MAX+5];
void dfs(int nd) { vis[nd] = true;
  while(ed[nd].size()) { int v = ed[nd].back();
    ed[nd].pop_back(); dfs(v);
  } sltn.pb(nd); }
// 0 -> nothing, 1 -> Trail, 2 -> Circuit exists
```

```
int findEuler (int n) {
  int src , snk , ret = 1;
  bool found_src = false, found_snk = false;
  CLR(inDeg); CLR(outDeg);
  for(int u = 1; u <= n; u++) {
    for(int i = 0; i<ed[u].size(); i++) {
      int v = ed[u][i]; out[u]++; in[v]++;}}
  int diff;
  for(int i = 1; i<=n; i++) {
    diff = out[i] - in[i];
    if(diff == 1) { if(found_src) return 0;
      found_src = true; src = i; }
    else if (diff == -1) {
      if(found_snk) return 0;
      found_snk = true; snk = i; }
    else if(diff != 0) return 0; }
  if(!found_src) { ret = 2;
    for(int i = 1 ; i <= n ; i++) {
      if( out[i] ) {
        found_src = true; src = i; break;}}}
  if(!found_src) return ret;
  CLR(vis); sltn.clear(); dfs(src);
  for(int i = 1; i<=n; i++)
    if(out[i] && !vis[i]) return 0;
  for(int i = (int)sltn.size()-1; i>=0; i--) printf("%d ",sltn[i]);
  return ret; }
```

1.20 FFT

```
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 operator / (const ld &x) const {return cplx(a / x, b / x);}
  const cplx conj() const {return cplx(a, -b);}
};
const ld PI = acosl(-1); 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) {
    cplx w_n = cplx(cosl(2 * PI / (1 << (sz + 1))), sinl(2 * PI / (1 << (sz + 1))));
    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];
vector <ll> multiply (vector <ll> a, vector <ll> b) {
  int n = a.size(), m = b.size(), sz = 1;
  while (sz < n + m - 1) sz <<= 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); vector <ll> c(n + m - 1);
  for (int i = 0; i < n + m - 1; ++i) c[i] = f[i].a / sz + 0.3;
  return c;
}
```

1.21 FWHT

```
const int OR = 0; const int AND = 1; const int XOR = 2; const int N = (1 << 20) + 5;
namespace FWHT { ll a[N], b[N];
void forward_fwht (ll *arr, int n, int flag = XOR) {
  if (n == 0) return;
  int i, m = n >> 1; forward_fwht(arr, m, flag); forward_fwht(arr + m, m, flag);
  // apply mod if required
  for (i = 0; i < m; ++i) {
    ll x = arr[i], y = arr[i + m];
    if (flag == OR) arr[i] = x, arr[i + m] = x + y;
    if (flag == AND) arr[i] = x + y, arr[i + m] = y;
    if (flag == XOR) arr[i] = x + y, arr[i + m] = x - y;
  }
}
void inverse_fwht (ll *arr, int n, int flag = XOR) {
  if (n == 0) return; int i, m = n >> 1; inverse_fwht(arr, m, flag); inverse_fwht(arr + m, m, flag);
  // apply mod if required
  for (i = 0; i < m; ++i) {
    ll x = arr[i], y = arr[i + m];
    if (flag == OR) arr[i] = x, arr[i + m] = y - x;
    if (flag == AND) arr[i] = x - y, arr[i + m] = y;
    if (flag == XOR) arr[i] = (x + y) >> 1, arr[i + m] = (x - y) >> 1;
  }
}
vector <ll> convolution (int n, ll *A, ll *B, int flag = XOR) {
  assert(!(n & (n - 1))); for (int i = 0; i < n; ++i) a[i] = A[i]; for (int i = 0; i < n; ++i) b[i] = B[i];
  forward_fwht(a, n, flag); forward_fwht(b, n, flag);
  // apply mod if required
  for (int i = 0; i < n; ++i) a[i] = a[i] * b[i]; inverse_fwht(a, n, flag); return vector <ll> (a, a + n);
}
}
```

1.22 Floor Sum of Arithmatic Progression

```
// sum [(x + kn) / m] for 0 <= k < m
inline ll aux (ll x, ll n, ll m) { ll g = __gcd(n, m); return g * (x / g) + (m * n - m - n + g) / 2; }
// sum [(x + kn) / m] for 0 <= k < lim < m
ll get (ll x, ll n, ll m, ll lim) {
```

```
  if (!lim) return 0;
  ll ret = lim * (x / m) + lim * (lim - 1) * (n / m) / 2;
  n %= m, x %= m; if (!n) return ret;
  ll p = (x + (lim - 1) * n) / m;
  return ret + p * lim - get(m - x + n - 1, m, n, p); }
// sum [(x + kn) / m] for 0 <= k < lim in O(lg max(n, m)), m > 0, lim >= 0
inline ll floorAPsum (ll x, ll n, ll m, ll lim) {
  ll ret = 0; if (x < 0) {
    ll q = x / m; x %= m;
    if (x) x += m, --q;
    ret += q * lim;
  } if (n < 0) {
    ll q = n / m; n %= m;
    if (n) n += m, --q;
    ret += lim * (lim - 1) * q / 2;
  } ll tmp = aux(x, n, m), tot = lim / m;
  ret += tmp * tot + tot * (tot - 1) * n * m / 2 + tot * n * (lim - tot * m);
  return ret + get(x, n, m, lim % m); }
```

1.23 Gauss-Jordan(Mod 2)

```
const int SZ = 105; const int MOD = 1e9 + 7;
bitset <SZ> mat[SZ]; int where[SZ]; bitset <SZ> ans;
/// n for row, m for column, modulo 2
int GaussJordan(int n,int m) {
  SET(where); /// sets to -1
  for(int r=0,c=0; c<m && r<n; c++) {
      for(int i=r; i<n; i++) if( mat[i][c] ) { swap(mat[i],mat[r]); break; }
      if( !mat[r][c] ) continue; where[c] = r;
      for (int i=0; i<n; ++i) if (i != r && mat[i][c]) mat[i] ^= mat[r]; r++;
  }
  for(int j=0; j<m; j++) {
      if(where[j]!=-1) ans[j] = mat[where[j]][m]/mat[where[j]][j];
      else ans[j] = 0; }
  for(int i=0; i<n; i++){
      int sum = 0; for(int j=0; j<m; j++) sum ^= (ans[j] & mat[i][j]);
      if( sum != mat[i][m] ) return 0; /// no solution
  }
  int cnt = 0; for(int j=0; j<m; j++) if (where[j]==-1) cnt++;
  return bigMod(2,cnt,MOD); /// how many solutions modulo some other MOD
}
```

1.24 Gaussian Related Problems

```
/***
Problem 1 : You are given an array a[0...(n-1)] of integer numbers. Your task is to divide the array into the maximum number of non empty segments in such a way that : there doesn't exist a non-empty subset of segments such that the XOR-sum of the numbers from them is equal to 0.
Build a cumilitive xor array. ( cum[i] = cum[i-1] ^ a[i] ) Build a matrix where row[i] = binary representation of cum[i] Answer is the rank of the matrix.
If cum[n-1] = 0, no solution.
***/
/*** Problem 2 : Given n numbers, you have to take a subset. Let X denote the xor of the numbers of the subset.
You will be given some conditions on the bits of X. condition(b, v) -> try to make the b'th bit of X v(0/1) The conditions will be ordered
Suppose 6 conditions are given. Y = 100110 denotes condition 1,4,5 are satisfied. We will try to maximize Y.
***/
const int MAX = 100010;
ll ara[MAX]; bitset <MAX> mat[70];
int n, row, ans[MAX], where[MAX];
void addCondition(int bn,int val){
  ++row; mat[row].reset(); mat[row][n] = val;
  for(int col=0; col<n; col++)
    mat[row][col]=( (ara[col]>>bn) & 1 );
  for(int col=0; col<n; col++){
    if(mat[row][col]){
      if(where[col]) mat[row] ^= mat[where[col]];
      else break; }}
  for(int col=0; col<n; col++){
    if(mat[row][col]){ where[col]=row; return; }} --row;
}
struct data { int bitNumber; int val; // preferred value for that bit };
vector <data> conditions;
/// m denotes maximum number of bits of any number in the input
void solve() { CLR(where); row = 0;
  for(int i=0;i<conditions.size();i++){
    addCondition(conditions[i].bitNumber, conditions[i].val);
  }
  for(int i=n-1;i>=0;i--){
    if(mat[where[i]][n]){ ans[i] = 1;
      for(int j=1;j<=row;j++){
        if(mat[j][i]) mat[j].flip(n);
    }}else ans[i] = 0;}}
int main() { // scan n integer numbers, fill conditions vector
  solve(); // ans[i] will be 1 if the i'th integer is taken }
/*** Problem 3: Given an array ara[] of n integers, you will be given some queries. ? L x --> How many subsequences of ara[1:L] has XOR-sum = x
Let's suppose we have the basis vectors for the elements upto L. ans is 0 if, x is not representable by the basis vectors. Otherwise, ans for any query upto L is = 2 ^ (L-S),
S -> size of basis ***/
```

1.25 Gaussian-Jordan

```
double mat[SZ][SZ], ans[SZ]; int where[SZ];
int GaussJordan(int n,int m) {
  SET(where); /// sets to -1
  for(int r=0,c=0; c<m && r<n; c++) {
    int mx = r; for(int i=r; i<n; i++)
        if( abs(mat[i][c]) > abs(mat[mx][c]) ) mx = i;
    if( abs(mat[mx][c]) < EPS ) continue;
    if(r != mx) for(int j=c; j<=m; j++) swap(mat[r][j],mat[mx][j]);
    where[c] = r;
    for(int i=0; i<n; i++) if( i!=r ) {
        double mul = mat[i][c]/mat[r][c];
        for(int j=c; j<=m; j++) mat[i][j] -= mul*mat[r][j]; } r++; }
  for(int j=0; j<m; j++) {
      if(where[j]!=-1) ans[j] = mat[where[j]][m]/mat[where[j]][j];
      else ans[j] = 0; }
  for(int i=0; i<n; i++){
    double sum = 0;
```

```
    for(int j=0; j<m; j++) sum += ans[j] * mat[i][j];
    if( abs(sum - mat[i][m]) > EPS )
      return 0; /// no solution }
  for(int j=0; j<m; j++) if (where[j]==-1) return INF;
  return 1;
}
```

1.26 HLD

```
FenwickTree f[C]; vector <int> g[N]; int t, n, m, ptr, c[N], par[N]; int sz[N], h[N], in[N], nxt[N];
void dfs (int u = 1, int far = 0) { sz[u] = 1, h[u] = far;
  for (int v : g[u])
    g[v].erase(find(g[v].begin(), g[v].end(), u));
  for (int &v : g[u]) {
    par[v] = u; dfs(v, far + 1); sz[u] += sz[v];
    if (sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
  }}
void hld (int u = 1) { in[u] = ++ptr;
  for (int v : g[u]) {
    nxt[v] = (v == g[u][0] ? nxt[u] : v); hld(v);}}
int query (int col, int u, int v) {
  int res = 0;
  while (nxt[u] != nxt[v]) {
    if (h[nxt[u]] > h[nxt[v]]) swap(u, v);
    res += f[col].query(in[nxt[v]], in[v]);
    v = par[nxt[v]];
  } if (h[u] > h[v]) swap(u, v); res += f[col].query(in[u], in[v]); return res; }
int main() { ptr = 0; dfs(); hld(); }
```

1.27 HopcroftKarp

```
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() {
  // Maximum Matching, Minimum Vertex Cover
  int ans = hopcroftKarp();
  // Maximum Independent Set
  int offset = n - ans;
  cout << ans << " " << offset << '\n'; }
```

1.28 Hungarian Algorithm

```
#define MAXIMIZE -1
#define MINIMIZE +1
namespace wm{
using T = int; const T INF = ?; const int MAX = ?;
bool vis[MAX]; int P[MAX], way[MAX], match[MAX];
T U[MAX], V[MAX], minv[MAX], ara[MAX][MAX];
/// n = number of row and m = number of columns in 1 based, flag = MAXIMIZE or MINIMIZE
T hungarian(int n, int m, T mat[MAX][MAX], int flag){
  CLR(U), CLR(V), CLR(P), CLR(ara), CLR(way);
  for (int i = 1; i <= n; i++){
    for (int j = 1; j <= m; j++){
      ara[i][j] = flag * mat[i][j]; } }
  if (n > m) m = n; int a, b, d; T r, w;
  for (int i = 1; i <= n; i++){
    P[0] = i, b = 0;
    for (int j = 0; j <= m; j++) minv[j] = INF, vis[j] = false;
    do{ vis[b] = true; a = P[b], d = 0, w = INF;
      for (int j = 1; j <= m; j++){
        if (!vis[j]){
          r = ara[a][j] - U[a] - V[j];
          if (r < minv[j]) minv[j] = r, way[j] = b;
          if (minv[j] < w) w = minv[j], d = j; }}
      for (int j = 0; j <= m; j++){
        if (vis[j]) U[P[j]] += w, V[j] -= w;
        else minv[j] -= w; } b = d;
    } while (P[b] != 0);
    do{ d = way[b]; P[b] = P[d], b = d;
    } while (b != 0); }
  for (int j = 1; j <= m; j++)
    match[P[j]] = j;
  return (flag == MINIMIZE) ? -V[0] : V[0];
}
}
```

1.29 KMP and Z Algorithm

```
char str[MAX]; int pref[MAX];
void prefixFunction(int P) { int j=0;
  for(int i = 1; i < P; i++) {
    while(true) { if(str[i] == str[j]) {
        j = pref[i] = j+1; break; }
      else { if(j==0) { pref[i] = 0; break; }
        else j = pref[j-1]; }}}}

// z algorithm
char s[N];
int t, n, z[N];
int main() {
  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; }}}}
```

1.30 KnuthOpt

```
// optimal way to break a string at positions x1, x2, ..., xk
// so that cost of a break = length of the string
```

```
// positions: 3, 8, 10
// thisisastringofchars     (original)
// thi sisastringofchars    (cost:20 units)
// thi sisas tringofchars   (cost:17 units)
// thi sisas tr ingofchars  (cost:12 units)
//                          Total: 49 units.
// optimal order to break it with least cost
// dp(i, j) = min_{i < k < j} dp(i, k) + dp(k, j) + cost(i, j)
// opt[i][j-1] <= opt[i][j] <= opt[i+1][j]
// so apply knuth opt
for (int s = 0; s<=k; s++) //s - length(size) of substring
  for (int l = 0; l+s<=k; l++) { //l - left point
    int r = l + s;   //r - right point
    if (s < 2) {
      res[l][r] = 0; //DP base - nothing to break
      mid[l][r] = l; //mid is equal to left border
      continue;
    }
    int mleft = mid[l][r-1]; //Knuth's trick: getting bounds on m
    int mright = mid[l+1][r];
    res[l][r] = 1000000000000000000LL;
    for (int m = mleft; m<=mright; m++) {
    //iterating for m in the bounds only
      int64 tres = res[l][m] + res[m][r] + (x[r]-x[l]);
      if (res[l][r] > tres) {    //relax current solution
        res[l][r] = tres;
        mid[l][r] = m;
      }
    }
  }
int64 answer = res[0][k];
```

1.31 Lichao Tree

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

1.32 MCMF(SPFA)

```
namespace mcmf {
using T = int; const T INF = ?; const int MAX = ?;
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,int cap,int cost){
  Edge a = {v,ed[v].size(),cap,cost,0};
  Edge b = {u,ed[u].size(),0,-cost,0};
  ed[u].pb(a); ed[v].pb(b); }
inline bool SPFA(){ CLR(vis);
  for(int i=1; i<=n; i++) mCap[i] = dis[i] = INF;
  queue <int> q; dis[src] = 0; vis[src] = true; q.push(src);
  while(!q.empty()){
    int u = q.front(); q.pop(); vis[u] = false;
    for(int i=0; i<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;
```

```cpp
    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 mp(F,C);
}}
```

1.33 Matrix Tree

Let A be the adjacency matrix of the graph: A[u][v] is the number of edges between u and v.

Let D be the degree matrix of the graph: a diagonal matrix with D[u][u] being the degree of vertex u (including multiple edges, ignore self loops).

The Laplacian matrix of the graph is defined as L = D - A.
M -> Submatrix of L discarding the last row and last column([n-1] * [n-1])

The number of different spanning tress of the graph = Determinant of M.

1.34 Maximum Points to Enclose in a Circle of Given Radius with Angular Sweep

```cpp
typedef pair<double,bool> pdb;
#define START 0
#define END 1
struct PT
{
 double x, y;
 PT() {}
 PT(double x, double y) : x(x), y(y) {}
 PT(const PT &p) : x(p.x), y(p.y)    {}
 PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
 PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
 PT operator * (double c)     const { return PT(x*c,   y*c  ); }
 PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};
PT p[505];
double dist[505][505];
int n, m;
void calcDist()
{
 FOR(i,0,n)
 {
  FOR(j,i+1,n)
   dist[i][j]=dist[j][i]=sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)+(p[i].y-p[j].y)*(p[i].y-p[j].y));
 }
}
// Returns maximum number of points enclosed by a circle of radius 'radius'
// where the circle is pivoted on point 'point'
// 'point' is on the circumfurence of the circle
int intelInside(int point, double radius) {
 vector<pdb> ranges;
 FOR(j,0,n) {
  if(j==point || dist[j][point]>2*radius) continue;
  double a1=atan2(p[point].y-p[j].y,p[point].x-p[j].x);
  double a2=acos(dist[point][j]/(2*radius));
  ranges.pb({a1-a2,START});
  ranges.pb({a1+a2,END});
 }
 sort(ALL(ranges));
 int cnt=1, ret=cnt;
 for(auto it: ranges) {
  if(it.second) cnt--;
  else cnt++;
  ret=max(ret,cnt);
 } return ret;
}
// returns maximum amount of points enclosed by the circle of radius r
// Complexity: O(n^2*log(n))
int go(double r)  {
 int cnt=0;
 FOR(i,0,n) cnt=max(cnt,intelInside(i,r));
 return cnt; }
```

1.35 MinEnclosingCircle

```cpp
#include <bits/stdc++.h>
typedef pair <ld, ld> point;
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;
}
```

1.36 ModMul, ModLog, ModSqrt

```
typedef unsigned long long ull;
typedef long double ld;
ull mod_mul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(ld(a) * ld(b)
      / ld(M));
  return ret + M * (ret < 0) - M * (ret
      >= (ll)M);
}
// Returns the smallest $x \ge 0$ s.t.
    $a^x = b \pmod m$. a and m must be
    coprime.
// Time: $O(\sqrt m)$
ll modLog(ll a, ll b, ll m) {
  assert(__gcd(a, m) == 1);
  ll n = (ll) sqrt(m) + 1, e = 1, x = 1,
      res = LLONG_MAX;
  unordered_map<ll, ll> f;
  rep(i,0,n) e = e * a % m;
  rep(i,0,n) x = x * e % m, f.emplace(x,
      i + 1);
  rep(i,0,n) if (f.count(b = b * a % m))
    res = min(res, f[b] * n - i - 1);
  return res;
}
// mod_sqrt
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p; if (a == 0)
      return 0;
  assert(modpow(a, (p-1)/2, p) == 1); //
      else no solution
  if (p % 4 == 3) return modpow(a,
      (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4
      works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0) ++r, s /= 2;
  /// find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p -
      1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n,
      s, p);
  for (;; r = m) { ll t = b;
    for (m = 0; m < r && t != 1; ++m) t =
        t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1),
        p);
    g = gs * gs % p; x = x * gs % p; b =
        b * g % p;}}
```

1.37 Monotonous Set

```
// insert function inserts a pair(x,y)
    into the structure, query(v) returns
    the maximum value y such that x <= v
    and
struct MonotonousSet{ set < pii > S;
void insert(pii p){ S.insert(p);
  auto it = S.find(p);
  if(it != S.begin()){ auto tmp = it;
    --tmp; if(tmp->yy >= it->yy){
      S.erase(it); return; }} ++it;
  while(it!=S.end() && it->yy<=p.yy){
    S.erase(it); it = S.find(p); ++it; }}
int query(int v){ if(S.empty()) return 0;
  auto it = S.upper_bound({v,INF});
  if(it==S.begin()) return 0;
  return (--it)->second; }
void clear() { S.clear();}};
```

1.38 NTT

```
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;
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)
// Check if a G works: does not work iff
    G^((MOD-1)/p) = 1 (mod MOD)
// for some prime factor p of MOD-1
```

1.39 Non-Linear-Recurrence

```
non linear reccurence
f(x) = kx + axf(x) + bf(x)^2
let g = f(x),
then, bf(x)^2 + (ax - 1)f(x) + kx = 0
----> bg^2 + (ax - 1)g + kx = 0
----> g = (1 - ax -+ sqrt(a^2x^2 - (2a +
    4kb)x + 1)) / (2b)
```

```
suppose, Q^2 = (a^2x^2 - (2a + 4kb)x + 1) ----(i)
then 2bg = 1 - ax - Q
---> Q = 1 - ax - 2bg
---> Q1 = -a - 2bg1 ----(ii)

from (i)
2QQ1 = 2a^2x - 2a - 4kb
QQ1 = a^2x - a - 2kb
Q(-a - 2bg1) = a^2x - a - 2kb [from--->ii]
Q(a + 2bg1) = a + 2kb - a^2x
Q^2 (a + 2bg1) = Q(a + 2kb - a^2x)
(a + 2bg1)(a^2x^2 - (2a + 4kb)x + 1) = (1 - ax - 2bg)(a + 2kb - a^2x)
```

1.40 Number of Arrays Having Non Equal Consecutive Elements

```cpp
// Number of arrays of size n having the first element as 1,
// the last element x and all the other elements between
// [1,k] and no two consecutive elements are equal

// If k changes, preprocess has to be called
// Preprocess -> O(n), Complexity -> O(1)
// If no bound on the first and last element, then there are k * (k-1)^(n-1) arrays
int dp[MAX];
void preprocess(int n, int k) {
  dp[0] = 0; dp[1] = 1;
  for (int i=2;i<n;i++) {
    dp[i] = ( (k - 2) * 1LL * dp[i - 1] ) % MOD;
    dp[i] += ( (k - 1) * 1LL * dp[i - 2] ) % MOD;
    if(dp[i] >= MOD) dp[i] -= MOD;
  }}
int howMany(int n,int k,int x) {
    if(x == 1) return ( (k - 1) * 1LL * dp[n - 2] ) % MOD;
    else return dp[n-1];
}
```

1.41 Online Bridge

```cpp
// adds and edge to the ds, outputs the number of bridges currently
const int N = 1e5+5;
int n, bridges, m, current = 0;
int bcc[N], comp[N], link[N], sz[N], vis[N];
void init() {
 for(int i=0;i<=n;i++) {
  bcc[i] = comp[i] = i; link[i] = -1; sz[i] = 1;
 } bridges = 0; }
int getBCC(int u) { if(u == -1) return -1;
 if(bcc[u] == u) return u;
 return bcc[u] = getBCC(bcc[u]); }
int getComp(int u) {
 if(comp[u] == u) return u;
 return comp[u] = getComp(comp[u]); }
void mergePath(int u, int v) {
 current++; vector<int> va, vb;
 int lca = -1;
 while(lca == -1) {
  if(u != -1) { u = getBCC(u);
   va.push_back(u);
   if(vis[u] == current) lca = u;
   vis[u] = current; u = link[u]; }
  if(v != -1) {
   vb.push_back(v); v = getBCC(v);
   if(vis[v] == current) lca = v;
   vis[v] = current; v = link[v]; }}
 for(auto &it:va) {
  bcc[it] = lca; if(it == lca) break; bridges--; }
 for(auto &it:vb) {
  bcc[it] = lca; if(it == lca) break; bridges--; }}
void MakeRoot(int u) {
 u = getBCC(u); int root = u, child = -1;
 while(u != -1) {
  int par = getBCC(link[u]);
  link[u] = child; comp[u] = root;
  child = u; u = par; }
 sz[root] = sz[child]; }
void addEdge(int u, int v) {
 u = getBCC(u), v = getBCC(v);
 if(u == v) return;
 int compu = getComp(u), compv = getComp(v);
 if(compu != compv) { bridges++;
  if(sz[compu] > sz[compv]) {
   swap(u, v); swap(compu, compv); }
  MakeRoot(u); link[u] = v; comp[u] = v;
  sz[compv] += sz[compu]; }
 else mergePath(u, v);
}
int32_t main() {
 int t; cin>>t; while(t--) {
  cin>>n>>m; init();
  for(int i=0;i<m;i++) {
   int u, v; cin>>u>>v; addEdge(u, v);
   cout<<bridges<<endl; } }
}
```

1.42 Online FFT

```cpp
int n, G[SZ], F[SZ], dp[SZ];
bool vis[SZ]; vector <int> segmentation(int l1, int r1, int l2, int r2){
  int i, j; vector <int> a, b;
     a.resize(r1 - l1 + 1); b.resize(r2 - l2 + 1);
  for(i = l1, j = 0; i <= n && i <= r1; i++, j++) { a[j] = F[i]; }
  for(i = l2, j = 0; i <= n && i <= r2; i++, j++) { b[j] = G[i]; } return multiply(a, b);
}
void contribute(int offst, vector <int> &poly){
  int i, j; for(j = 0; j < poly.size() && j + offst <= n; j++){
     F[j + offst] = add(F[j + offst], poly[j]); }}
void solve(){ int i, j, l, ret; F[0] = 1;
    for(i = 0; i <= n; i++){ F[i] = G[i]; }
    for(i = 0; i <= n; i++){
      for(j = 0; (1 << j) <= i + 1; j++){
        if((i + 1) % (1 << j) == 0){
          vector <int> a = segmentation(i - (1 << j) + 1, i, 1 << j, (1 << j) + (1 << j) - 1);
          int offst = (1 << j) * (i / (1 << j) + 1);
          contribute(offst, a);
          para(l, 0, n, F);}}}
    dp[0] = 1;
    for(i = 0; i <= n; i++) ret = brute(i);
    para(i, 0, n, dp); para(i, 0, n, F);
}
```

1.43 Palindromic Tree

```cpp
// depth[nd] = Number of suffixes of P that are palindrome
// fin[i] = Number of palindromes that end at index i
namespace pt {
const int N = 100010; const int C = 26;
char str[N];
```

```
int len[N], link[N], ed[N][C], occ[N],
  st[N];
int fin[N], depth[N], n, nc, suff, pos;
void init() {
  str[0] = -1; nc = 2; suff = 2;
  len[1] = -1, link[1] = 1, len[2] =  0,
    link[2] = 1;
  CLR(ed[1]), CLR(ed[2]); occ[1] = occ[2]
    = 0;
}
inline int scale(char c) { return c-'a'; }
inline int nextLink(int cur) {
  while (str[pos - 1 - len[cur]] !=
    str[pos]) cur = link[cur];
  return cur;
}
inline bool addLetter(int p) { pos = p;
  int let = scale(str[pos]); int cur =
    nextLink(suff);
  if (ed[cur][let]) {
    suff = ed[cur][let]; fin[pos] =
      depth[suff];
    occ[suff]++; return false;
  }
  suff = ++nc; CLR(ed[nc]); len[nc] =
    len[cur] + 2;
  ed[cur][let] = nc; occ[nc] = 1;
  if (len[nc] == 1) {
    st[nc] = pos; link[nc] = 2;
    fin[pos] = depth[nc] = 1; return true;
  }
  link[nc] = ed[nextLink(link[cur])][let];
  fin[pos] = depth[nc] = depth[link[nc]]
    + 1;
  st[nc] = pos-len[nc] + 1;
  return true;
}
void build(int _n) {
  n = _n; init(); for(int i=1;i<=n;i++)
    addLetter(i);
  for(int i=nc;i>=3;i--) occ[link[i]] +=
    occ[i];
  occ[2] = occ[1] = 0;
}
}
scanf("%s",pt::str+1);
  pt::build(strlen(pt::str+1));
```

1.44 Point in Polygon Binary Search

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

1.45 Pollard Rho Miller Rabin

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return n -
    2 < 2;
  ull A[] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022}, s =
    __builtin_ctzll(n-1), d = n >> s;
  for(auto &a : A) {
    ull p = mod_pow(a, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n
      && i--) p = mod_mul(p, p, n);
    if (p != n-1 && i != s) return 0;
  } return 1; }
ull pollard(ull n) {
  auto f = [n](ull x) { return
    (mod_mul(x, x, n) + 1) % n; };
  if (!(n & 1)) return 2;
  for (ull i = 2;; i++) {
    ull x = i, y = f(x), p;
    while ((p = __gcd(n + y - x, n)) ==
      1) x = f(x), y = f(f(y));
    if (p != n) return p; }}
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(), all(r)); return l; }
```

1.46 RMQ 2D

```
template<class num_t, class cmp =
  less<num_t> >
struct RMQ2D {
static const int maxn = 1e3 + 5;
static const int maxm = 1e3 + 5;
static const int logn = 10 + 1;
static const int logm = 10 + 1;
int n, m; num_t a[maxn][maxm];
num_t f[logm][maxn][maxm];
num_t g[logm][logn][maxm][maxn];
inline num_t best(const num_t& a, const
    num_t& b) { return cmp()(a, b) ? a :
    b; }
void init(int _n, int _m) { n = _n, m =
  _m; }
num_t* operator [] (int u) { assert(u <
  n); return a[u]; }
void build() {
  for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= m; i++) {
      f[0][k][i] = a[k - 1][i - 1]; }
    for (int j = 1; 1 << j <= m; j++) {
      for (int i = 0; i + (1 << j) - 1 <=
        m; i++) {
```

```
        f[j][k][i] = best(f[j - 1][k][i],
          f[j - 1][k][i + (1 << (j - 1))]); }}}
  for (int k = 1; k <= m; k++) {
    for (int l = 0; k + (1 << l) - 1 <= m; l++) {
      for (int i = 1; i <= n; i++) {
        g[l][0][k][i] = f[l][i][k]; }
      for (int j = 1; 1 << j <= n; j++) {
        for (int i = 0; i + (1 << j) - 1 <= n; i++) {
          g[l][j][k][i] = best(g[l][j -
            1][k][i], g[l][j - 1][k][i + (1 << (j - 1))]); }}}}}
num_t query(int x, int y, int z, int t) {
  x++, y++, z++, t++;
  int a = z - x + 1, b = t - y + 1;
  int lga = __lg(a), lgb = __lg(b);
  int res = g[lgb][lga][y][x];
  res = best(res, g[lgb][lga][y + b - (1 << (lgb))][x + a - (1 << (lga))]);
  res = best(res, g[lgb][lga][y][x + a - (1 << (lga))]);
  res = best(res, g[lgb][lga][y + b - (1 << (lgb))][x]);
  return res; }
}; RMQ2D<int> rmq;
int main() {
  int n, m, x, q; cin >> n >> m; rmq.init(n, m);
  for(int i=0;i<n;i++) for(int j=0;j<m;j++) cin >> x, rmq[i][j] = x;
  rmq.build(); cin >> q; int a, b, c, d;
  for(int i=0;i<q;i++) {
    cin >> a >> b >> c >> d;
    // int mn = INF; for(int i=a;i<=c;i++) for(int j=b;j<=d;j++) mn = min(mn, rmq[i][j]);
    cout << rmq.query(a, b, c, d) << endl; // returns mn
  }
  return 0;
}
```

1.47 Rope, GP, PBDS

```
#include <ext/pb_ds/assoc_container.hpp> #include <ext/pb_ds/tree_policy.hpp> using namespace __gnu_pbds;
typedef tree<int, null_type, less <int>,
    rb_tree_tag,
    tree_order_statistics_node_update >
    ordered_set; // less_equal for multiset
// OS.find_by_order(x) returns iterator, OS.order_of_key(x) return value
#include <ext/rope> using namespace __gnu_cxx;
rope <char> R;
// R.push_back(x) inserts character x at the end of rope R
// R.insert(pos, nr) inserts rope nr into R at position pos (the first character of nr will be in position pos)
// R.erase(pos, cnt) deletes segment [pos, pos+cnt-1] from R
// R.substr(pos, cnt) = segment [pos, pos+cnt-1], returns a rope
// for(rope <char>::iterator it = R.mutable_begin(); it != R.mutable_end(); ++it) cout << *it;
#include <ext/pb_ds/assoc_container.hpp> using namespace __gnu_pbds;
gp_hash_table <int, int> table;
const int RANDOM =
    chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash { int operator()(int x) const { return x ^ RANDOM; } };
gp_hash_table<int, int, chash> table;
struct chash { int operator()(pii x) const { return x.first * 31 + x.second;}};
gp_hash_table<pii, int, chash> table;
```

1.48 SOS DP

```
// O(3^N)
for (int mask = 0; mask < (1<<n); mask++){
  F[mask] = A[0];
  for(int submask = mask; submask > 0; submask = (submask-1) & mask) {
    F[mask] += A[i];}}
// O(N * 2^N)
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i)
  for(int mask = 0; mask < (1<<N); ++mask){
    if(mask & (1<<i))
      F[mask] += F[mask^(1<<i)];
  }
// F[mask] = sum of A[i] given than (i & mask) = mask
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i)
  for(int mask = (1<<N)-1; mask >= 0; --mask){
    if (!(mask & (1<<i)))
      F[mask] += F[mask | (1<<i)];
  }
/// How many pairs in ara[] such that (ara[i] & ara[j]) = 0
/// N --> Max number of bits of any array element
const int N = 20; int inv = (1<<N) - 1;
int F[(1<<N) + 10]; int ara[MAX];
/// ara is 0 based
ll howManyZeroPairs(int n,int ara[]) {
  CLR(F); for(int i=0;i<n;i++) F[ara[i]]++;
  for(int i = 0;i < N; ++i)
    for(int mask = 0; mask < (1<<N); ++mask){
      if(mask & (1<<i))
        F[mask] += F[mask^(1<<i)];
    } ll ans = 0;
  for(int i=0;i<n;i++) ans += F[ara[i] ^ inv]; return ans;}
/// Number of subsequences of ara[0:n-1] such that
/// sub[0] & sub[2] & ... & sub[k-1] = 0
///0 based array
const int N = 20; int inv = (1<<N) - 1;
int F[(1<<N) + 10]; int ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
int howManyZeroSubSequences(int n,int ara[]) {
  CLR(F); for(int i=0;i<n;i++) F[ara[i]]++;
  for(int i = 0;i < N; ++i)
    for(int mask = (1<<N)-1; mask >= 0; --mask){
      if (!(mask & (1<<i)))
        F[mask] += F[mask | (1<<i)];
    }int ans = 0;
  for(int mask=0;mask<(1<<N);mask++) {
    if(countBit(mask) & 1) ans -= p2[F[mask]];
    else ans += p2[F[mask]];
```

```
    /// p2[F[mask]] is the count of subsets that will have the mask on at least
    if(ans<0) ans += MOD;
    if(ans>=MOD) ans -= MOD; } return ans;}
/// Number of subsequences of ara[0:n-1] such that
/// sub[0] | sub[2] | ... | sub[k-1] = Q
int F[(1<<20) + 10], ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
/// ara is 0 based
int howManySubsequences(int n, int ara[], int m, int Q) {
  CLR(F); for(int i=0;i<n;i++) F[ara[i]]++;
  if(Q == 0) return sub(p2[F[0]], 1);
  for(int i = 0;i < m; ++i)
    for(int mask = 0; mask < (1<<m); ++mask){
      if (mask & (1<<i))
        F[mask] += F[mask ^ (1<<i)];
    }
  int ans = 0;
  for(int mask=0;mask<(1<<m);mask++) {
    if(mask & Q != mask) continue;
    if(__builtin_popcount(mask ^ Q) & 1) ans = sub(ans, p2[F[mask]]);
    else ans = add(ans, p2[F[mask]]);
  }
  return ans;
}
```

1.49 Sajib Centroid

```
vector < pair <int, char > > adj[SZ];
bool mark_cen[SZ];
int parent[SZ], sbtr[SZ], n, centroid, par_cen[SZ];
vector <int> adj_cen[SZ];
void dfs0(int src, int par){
   int i, j, u; sbtr[src] = 1, parent[src] = par;
   for0(i, adj[src].size()){
      u = adj[src][i].first;
      if(u != par){ dfs0(u, src), sbtr[src] += sbtr[u]; }
   }
}
int get_centroid(int src, int par){
   int i, j, u, mx = -1, bg;
   bool is_cen = true;
   for0(i, adj[src].size()){
      u = adj[src][i].first;
      if(u != par){
         if(sbtr[u] > n / 2){ is_cen = false; }
         if(sbtr[u] > mx){ mx = sbtr[u], bg = u; }
      }
   }
   if(is_cen && n - sbtr[src] <= n / 2){ return src; }
   return get_centroid(bg, src);
}
void get_centroid(int cnn, int root, int src, int par){
   int i, j, u, mx = -1, bg;
   bool is_cen = true;
   if(src == centroid){
      for0(i, adj[src].size()){
         u = adj[src][i].first;
         if(!mark_cen[u] && u != par){
            get_centroid(src, u, u, src); }
      }
      return;
   }
   for0(i, adj[src].size()){
      u = adj[src][i].first;
      if(u != par && !mark_cen[u]){
         if(sbtr[u] > sbtr[root] / 2){ is_cen = false; }
         if(sbtr[u] > mx){ mx = sbtr[u], bg = u; }
      }
   }
   if(mx == -1 || (is_cen && sbtr[root] - sbtr[src] <= sbtr[root] / 2)){
      adj_cen[cnn].push_back(src), adj_cen[src].push_back(cnn);
      mark_cen[src] = true;
      int p = parent[src];
      while(!mark_cen[p]){ sbtr[p] -= sbtr[src], p = parent[p]; }
      for0(i, adj[src].size()){
         u = adj[src][i].first;
         if(u != par && !mark_cen[u]){
            get_centroid(src, u, u, src); }
      }
      if(mx != -1 && !mark_cen[root]){
         get_centroid(src, root, root, parent[root]); }
   }
   else{ get_centroid(cnn, root, bg, src); }
}
void decompose(){
   int i, j;
   dfs0(0, -1), centroid = get_centroid(0, -1), dfs0(centroid, -1);
   for0(i, n + 2){ mark_cen[i] = false, adj_cen[i].clear(); }
   mark_cen[centroid] = true;
   get_centroid(centroid, centroid, centroid, -1);
   for0(i, n + 2){ mark_cen[i] = false; }
}
gp_hash_table < pair <int, int>, int, chash> mp[SZ], mp_child[SZ];
gp_hash_table < int, pair <int, int > > hash_mp[SZ];
gp_hash_table <int, int> dis_mp[SZ];
void dfs2(int cen, int src, int root, int par, int d, int h0, int h1, bool tick){
   int i, j, sz = adj_cen[src].size(), cnt = 0, nh0, nh1;
   int w, u;
   mp[cen][mpr(h0, h1)]++;
      dis_mp[cen][src] = d; hash_mp[cen][src] = mpr(h0, h1);
   if(tick){ mp_child[root][mpr(h0, h1)]++; }
   cnt = root == centroid ? 0 : 1;
   for0(i, adj[src].size()){
      u = adj[src][i].first;
      w = adj[src][i].second - 'a' + 1;
      nh0 = add(w, mul(h0, base[0], mod[0]), mod[0]), nh1 = add(w, mul(h1, base[1], mod[1]), mod[1]);
      if(!tick){
         if(u != par && !mark_cen[u]){
            dfs2(cen, u, adj_cen[src][cnt], src, d + 1, nh0, nh1, true); cnt++;
         }
         else if(u == par && !mark_cen[u]){
            dfs2(cen, u, adj_cen[src][sz - 1], src, d + 1, nh0, nh1, true);
         }
      }
      else if(u != par && !mark_cen[u]){
```

```
        dfs2(cen, u, root, src, d + 1,
          nh0, nh1, true);
      }
    }
}
void dfs1(int src, int par){
    int i, j, u; mark_cen[src] = true;
      par_cen[src] = par;
    mp[src].clear(); dis_mp[src].clear();
      hash_mp[src].clear();
    for0(i, adj_cen[src].size()){
        u = adj_cen[src][i];
        if(u != par && !mark_cen[u]){
          mp_child[u].clear();}
    }
    dfs2(src, src, src, parent[src], 0, 0,
      0, false);
    for0(i, adj_cen[src].size()){
        u = adj_cen[src][i];
        if(u != par && !mark_cen[u]){
          dfs1(u, src); }
    }
}
```

1.50 Smallest Enclosing Sphere

```
double enclosing_sphere(vector<double> x,
  vector<double> y, vector<double> z){
  int n = x.size();
  auto hyp = [](double x, double y,
    double z){
    return x * x + y * y + z * z; };
  double px = 0, py = 0, pz = 0;
  for(int i=0; i<n; i++){
    px += x[i]; py += y[i]; pz += z[i]; }
  px *= 1.0 / n; py *= 1.0 / n; pz *= 1.0
    / n;
  double rat = 0.1, maxv;
  int rounds = 7000; //should be as high
    as time limit allows
  for(int i=0; i<rounds; i++){
    maxv = -1;
    int maxp = -1;
    for(int j=0; j<n; j++){
      double tmp = hyp(x[j] - px, y[j] -
        py, z[j] - pz);
      if(maxv < tmp){
        maxv = tmp; maxp = j;
      }
    }
    px += (x[maxp] - px) * rat;
    py += (y[maxp] - py) * rat;
    pz += (z[maxp] - pz) * rat;
    rat *= 0.998; }
  printf("%.6lf %.6lf %.6lf\n", px, py,
    pz);
  return sqrt(maxv);
};
int main() {
  std::vector<double> v1, v2, v3;
  int n, x, y, z; cin >> n;
  for(int i = 0; i < n; i++) {
    cin >> x >> y >> z;
    v1.push_back(x); v2.push_back(y);
      v3.push_back(z); }
  enclosing_sphere(v1, v2, v3);
  return 0;
}
```

1.51 Space of Binary Vectors

```
// A vector can be added to the space at
  any moment
// Following queries can be made on the
  current basis at any moment
const int B = ?; struct space {
  int base[B], sz;
  void init() { CLR(base); sz = 0; }
  void add(int val) {
    for(int i = B-1; i >= 0; i--) {
      if( val & (1<<i) ) { if(!base[i]) {
          base[i] = val; ++sz; return;
        } else val ^= base[i]; } } }
  int getSize() { return sz; }
  // returns maximum possible ret such
    that, ret = (val ^ x)
  // and x is also in the vector space of
    the current basis
  int getMax(int val) { int ret = val;
    for(int i = B - 1; i >= 0; i--) {
      if(ret & (1<<i)) continue;
      ret ^= base[i]; } return ret; }
  // returns minimum possible ret such
    that, ret = (val ^ x)
  // and x is also in the vector space of
    the current basis
  int getMin(int val) { int ret = val;
    for(int i = B - 1; i >= 0; i--) {
      if( !(ret & (1<<i)) ) continue;
      ret ^= base[i]; } return ret; }
  // returns true if val is in the vector
    space
  bool possible(int val) {
    for(int i = B - 1; i >= 0; i--) {
      if(val & (1<<i)) val ^= base[i];
    } return (val == 0); }
  // returns the k'th element of the
    current vector space
  // if we sort all the elements
    according to their values
  int query(int k) {
    int ret = 0, tot = 1 << getSize();
    for(int i = B - 1; i >= 0; i--) {
      if(!base[i]) continue;
      int low = tot >> 1;
      if ((low < k && (ret & 1 << i) ==
          0) || (low >= k && (ret & 1 <<
          i) > 0))
        ret ^= base[i];
      if (low < k) k -= low; tot /= 2; }
        return ret; }
};
```

1.52 Spherical Distance

```
// Returns the shortest distance on the
    sphere with radius radius between the
    points with azimuthal angles
    (longitude) f1 (1) and f2 (2) from x
    axis and zenith angles (latitude) t1
    (1) and t2 (2) from z axis. 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);
}
```

1.53 Stanfordacm Geo Template

```
using namespace std;
double INF = 1e100; double EPS = 1e-12;
struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
```

```
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); } };
double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) { os << "(" << p.x << "," << p.y << ")";  }
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));  }
// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z, double a, double b, double c, double d) {
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {  return fabs(cross(b-a, c-d)) < EPS;  }
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d) &&
      fabs(cross(a-b, a-c)) < EPS &&
      fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2; c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}
// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) *
          (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  } return c; }
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
```

```cpp
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
// compute intersection of circle
 ↪ centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a,
 ↪ PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R))
   ↪ return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x -
     ↪ RotateCCW90(v)*y);
  return ret; }
// This code computes the area or
 ↪ centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates
 ↪ are listed in a clockwise or
// counterclockwise fashion.  Note that
 ↪ the centroid is often known as
// the "center of gravity" or "center of
 ↪ mass".
double ComputeSignedArea(const vector<PT>
 ↪ &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  } return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 *
   ↪ ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y -
     ↪ p[j].x*p[i].y);
  }
  return c / scale; }
// tests whether or not a given polygon
 ↪ (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j],
       ↪ p[k], p[l]))
        return false;
    }
  }
  return true;
}
int main() {  return 0; }
```

1.54 Suffix Array

```cpp
namespace sa { const int N = 100010;
char str[N]; int wa[N], wb[N], wv[N],
 ↪ wc[N];
int r[N], S[N], rnk[N], lcp[N];
int cmp(int *r, int a, int b, int l) {
  return r[a] == r[b] && r[a + l] == r[b
   ↪ + l]; }
void da(int *r, int *sa, int n, int m) {
  int i, j, p, *x = wa, *y = wb, *t;
  for(i = 0; i < m; i++) wc[i] = 0;
  for(i = 0; i < n; i++) wc[x[i] =
   ↪ r[i]]++;
  for(i = 1; i < m; i++) wc[i] += wc[i -
   ↪ 1];
  for(i = n - 1; i >= 0; i--)
   ↪ S[--wc[x[i]]] = i;
  for(j = 1, p = 1; p < n; j *= 2, m = p)
   ↪ {
    for(p = 0, i = n - j; i < n; i++)
     ↪ y[p++] = i;
    for(i = 0; i < n; i++) if(S[i] >= j)
     ↪ y[p++] = S[i] - j;
    for(i = 0; i < n; i++) wv[i] =
     ↪ x[y[i]];
    for(i = 0; i < m; i++) wc[i] = 0;
    for(i = 0; i < n; i++) wc[wv[i]] ++;
    for(i = 1; i < m; i++) wc[i] += wc[i
     ↪ - 1];
    for(i = n - 1; i >= 0; i--)
     ↪ S[--wc[wv[i]]] = y[i];
    for(t = x, x = y, y = t, p = 1,
     ↪ x[S[0]] = 0, i = 1; i < n; i++)
      x[S[i]]= cmp(y, S[i - 1], S[i], j)
       ↪ ? p - 1 : p++;
  }}
void calheight(int *r, int *sa, int n) {
  int i, j, k = 0; for(i = 1; i <= n;
   ↪ i++) rnk[S[i]] = i;
  for(i = 0; i < n; lcp[rnk[i++]] = k ) {
    for(k ? k-- : 0, j = S[rnk[i]-1];
     ↪ r[i+k] == r[j+k]; k++); }}
int f[__lg(N) + 2][N];
void lcp_rmq_init(int n) {
  for(int i = 0; i < n; i++) f[0][i] =
   ↪ lcp[i];
  for(int l = 0, k; (k = 1 << l) < n;
   ↪ l++) {
    for (int i = 0; i + k < n; i++) {
      f[l + 1][i] = min(f[l][i], f[l][i +
       ↪ k]);}}}
// returns the lcp of suffix a and suffix
 ↪ b
int lcp_query(int a, int b) {
  assert(a != b); // handle it locally
  a = rnk[a], b = rnk[b]; if(a > b)
   ↪ swap(a, b); a++;
  int l = a == b ? 0 : __lg(b - a);
  return min(f[l][a], f[l][b - (1 << l) +
   ↪ 1]); }
void build(int n) {
  for(int i = 0; str[i]; i++) r[i] =
   ↪ (int)str[i];
  r[n] = 0; da(r, S, n+1, 128);
  calheight(r, S, n); lcp_rmq_init(n +
   ↪ 1);}}
```

1.55 Suffix Automaton Sajib

```cpp
struct state{
    int len, link;
    map <char, int> next;
};
state st[SZ * 2];
int to_state = 0, last, f_occ[SZ * 2],
 ↪ d[2 * SZ], sbtr[2 * SZ], vrtx[2 * SZ];
int t = 0;
vector <int> adj[2 * SZ];
void sa_init(){
    to_state = 0, st[0].len = 0,
     ↪ st[0].link = -1;
    to_state++, last = 0;
}
void sa_extend(char c){
    int cur = to_state++;
    st[cur].len = st[last].len + 1,
     ↪ f_occ[cur] = st[cur].len - 1;
    st[cur].next.clear();
    int p = last;
    while(p != -1 && st[p].next.find(c)
     ↪ == st[p].next.end()){
        st[p].next[c] = cur;
        p = st[p].link;
```

```
    }
    if(p == -1){ st[cur].link = 0; }
    else{
        int q = st[p].next[c];
        if(st[p].len + 1 == st[q].len){
            st[cur].link = q; }
        else{
            int clone = to_state++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            f_occ[clone] = f_occ[q];
            while(p != -1 &&
              st[p].next[c] == q){
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link =
              clone;
        }
    } last = cur;
}
void dsu(int src, bool keep){
    int i, j, mx = -1, bg = -1, u, v;
    for0(i, adj[src].size()){
        u = adj[src][i];
        if(sbtr[u] > mx){
            mx = sbtr[u], bg = u;
        }
    }
    for0(i, adj[src].size()){
        u = adj[src][i];
        if(u != bg){
            dsu(u, false);
        }
    }
    if(bg != -1){ dsu(bg, true); }
    for0(i, adj[src].size()){
        u = adj[src][i];
        if(u != bg){
            for(j = d[u]; j <= d[u] +
              sbtr[u] - 1; j++){
                v = vrtx[j];
                if(f_occ[v] + 1 -
                  st[v].len == 0){
                    update(src, f_occ[v]);
                    end_pos.insert(f_occ[
                      v]);
                }
            }
        }
    }
    if(f_occ[src] + 1 - st[src].len == 0){
        update(src, f_occ[src]),
          end_pos.insert(f_occ[src]);
    }
    if(!keep){ end_pos.clear(); }
}
```

1.56 Suffix Automaton

```
// sub_i-> maximum substring that is
  endpos equivalent to node i
// cnt[i] = number of occurrences of
  sub_i in str
// terminal[i] = true, then sub_i is a
  suffix of str
// dp[i] = number of substrings that has
  sub_i as prefix                  *
namespace sa {
const int N = 100005 << 1; const int C =
  26;
char str[N];
int n, sz, last, len[N], link[N],
  ed[N][C], cnt[N];
bool terminal[N]; vector <int> G[N];
void init() { SET(ed[0]);
  len[0] = 0, link[0] = -1, sz = 1, last
    = 0, terminal[0] = false; }
inline int scale(char c) { return c -
  'a'; }
void extend(char c) {
  int cur = sz++; terminal[cur] = false;
    cnt[cur] = 1;
  SET(ed[cur]); len[cur] = len[last] + 1;
    int p = last;
  while (p != -1 && ed[p][c]==-1) {
    ed[p][c] = cur; p = link[p]; }
  if (p == -1) link[cur] = 0;
  else { int q = ed[p][c];
      if (len[p] + 1 == len[q]) link[cur]
        = q;
      else {
        int clone = sz++; len[clone] =
          len[p] + 1;
        memcpy(ed[clone],ed[q],sizeof(ed[
          q]));
        link[clone] = link[q];
        while (p != -1 && ed[p][c] == q) {
          ed[p][c] = clone; p = link[p]; }
        link[q] = link[cur] = clone;
        cnt[clone] = 0; terminal[clone] =
          false;
      }
  } last = cur;
}
void dfs(int s){ for(auto x : G[s])
  dfs(x), cnt[s] += cnt[x];}
ll dp[N];
ll call(int nd) {
  ll &ret = dp[nd]; int x;
  if(ret!=-1) return ret; ret = cnt[nd];
  for(int i=0; i<C; i++) {
      x = ed[nd][i]; if(x!=-1) ret +=
        call(x);
  } return ret;}
/// returns the lexicographically k'th
  substring of str
string lex_kth_substr(ll k) {
  if((k+k) > (n*(n+1LL))) return "No such
    line.";
  string ret = ""; int cur = 0, x;
  while(k>0) {
    for(int i=0; i<C; i++) {
      x = ed[cur][i]; if(x == -1)
        continue;
      if(call(x)>=k) {
        ret += (char)i + 'a'; cur = x; k
          -= cnt[x]; break;
      } k -= call(x); }} return ret;
}
void build() { init(); n = strlen(str);
  for(int i=0; i<n; i++)
    extend(scale(str[i]));
  for(int i=1; i<sz; i++)
    G[link[i]].pb(i);
  dfs(0);
  for(int i=0; i<sz; i++) G[i].clear();
  for(int i=last; i!=-1; i=link[i])
    terminal[i] = true;
  SET(dp);
}}
scanf("%s",sa::str); sa::build(n);
```

1.57 Treap

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef struct node {
  int prior, sz; ll val, sum, lazy;
  struct node *l, *r, *p; } node;
typedef node* pnode; pnode Treap;
inline int getSize(pnode t) { return t ?
  t->sz : 0; }
inline ll get_sum(pnode t) { return t ?
  t->sum : 0; }
inline void lazyUpdate(pnode t){
  if(!t or !t->lazy) return;
  t->val += t->lazy; t->sum += t->lazy *
    getSize(t);
```

```
  if(t->l) t->l->lazy += t->lazy;
  if(t->r) t->r->lazy += t->lazy;
  t->lazy=0; }
inline void operation(pnode t) {
  if(!t) return; lazyUpdate(t->l); lazyUpdate(t->r);
  t->sz = getSize(t->l) + 1 + getSize(t->r);
  t->sum = get_sum(t->l) + t->val + get_sum(t->r);
  if(t->l) t->l->p = t; if(t->r) t->r->p = t; }
void split(pnode t, pnode &l, pnode &r, int pos, int add = 0){
  if(!t) return void( l = r = NULL) ; lazyUpdate(t);
  int curr_pos = add + getSize(t->l)+1;
  if(curr_pos <= pos) split(t->r, t->r, r, pos, curr_pos), l = t;
  else split(t->l, l, t->l, pos, add), r = t; operation(t); }
void merge(pnode &t, pnode l, pnode r) {
  lazyUpdate(l); lazyUpdate(r);
  if(!l or !r) t = l ? l : r;
  else if(l->prior > r->prior) merge(l->r, l->r, r), t = l;
  else merge(r->l, l, r->l), t = r;
  operation(t); }
pnode newNode(ll val){
  pnode ret = (pnode)malloc(sizeof(node));
  ret->prior = rand(); ret->sz = 1;
  ret->val = ret->sum = val; ret->lazy = 0;
  ret->p = ret->l = ret->r = NULL; return ret; }
inline void point_update(pnode &t, int id, ll val) {
  int sz = getSize(t->l);
  if( sz == (id-1) ) {
    t->val = val; pnode cur = t;
    while(cur != NULL) operation(cur), cur = cur->p; }
  else if(sz < (id-1) ) point_update(t->r, id-sz-1, val);
  else point_update(t->l, id, val);
}
void Remove(pnode &t, int id){
  pnode L, mid, R, X; split(t, L, mid, id-1);
  split(mid, X, R, 1); delete X;
  merge(t, L, R); }
void Insert(pnode &t, int id, ll val){
  pnode L, R, mid; pnode it = newNode(val);
  split(t, L, R, id-1);
  merge(mid, L, it); merge(t, mid, R); }
/// add val to all the nodes [i:j]
void range_update(pnode t, int i, int j, ll val){
  pnode L, M, R;
  split(t, L, M, i-1); split(M, t, R, j-i+1);
  t->lazy += val; merge(M, L, t); merge(t, M, R); }
/// range query [i:j]
ll range_query(pnode t, int i, int j){
  pnode L, M, R;
  split(t, L, M, i-1); split(M, t, R, j-i+1);
  ll ans = t->sum; merge(M, L, t);
  merge(t, M, R); return ans; }
/// Freeing memory after each test case
void Delete(pnode &t){
  if(!t) return; if(t->l) Delete(t->l);
  if(t->r) Delete(t->r); delete(t); t = NULL; }
ll ara[11]; int main(){
  int n = 10;
  for(int i=1; i<=n; i++) merge(Treap,Treap,newNode(ara[i]));
  Delete(Treap); }
```

1.58 Triangle Area From Medians

```
double triAreaFromMedians( double ma, double mb, double mc){
  double x = 0.5 * ( ma + mb + mc );
  double a=x * ( x - ma ) * ( x - mb ) * ( x - mc );
  if( a < 0.0 ) return -1.0;
  else return sqrt( a ) * 4.0 / 3.0;
}
```

1.59 VirtualTree

```
vector <int> g[N], virt[N], cost[N];
int n, m, ptr, h[N], in[N], p[N][LG], stk[N];
void add_edge (int u, int v) { if (u == v) return; virt[u].emplace_back(v); virt[v].emplace_back(u); int w = abs(h[u] - h[v]); cost[u].emplace_back(w); cost[v].emplace_back(w); }
void buildTree (vector <int> &nodes) {
  if (nodes.size() <= 1) return;
  sort(nodes.begin(), nodes.end(), [] (int x, int y) {return in[x] < in[y];});
  int root = get_lca(nodes[0], nodes.back()), sz = nodes.size();
  ptr = 0, stk[ptr++] = root;
  for (int i = 0; i < sz; ++i) {
    int u = nodes[i], lca = get_lca(u, stk[ptr - 1]);
    if (lca == stk[ptr - 1]) {
      stk[ptr++] = u;
    } else {
      while (ptr > 1 and h[stk[ptr - 2]] >= h[lca]) { add_edge(stk[ptr - 2], stk[ptr - 1]), --ptr; }
      if (stk[ptr - 1] != lca) {
          add_edge(lca, stk[--ptr]);
          stk[ptr++] = lca, nodes.emplace_back(lca); }
      stk[ptr++] = u; }}
  if (find(nodes.begin(), nodes.end(), root) == nodes.end()) nodes.emplace_back(root);
  for (int j = 0; j + 1 < ptr; ++j) add_edge(stk[j], stk[j + 1]);
} int main() {
  // graph, dfs, lca
  cin >> m; vector <int> nodes(m);
  for (int i = 0; i < m; ++i) scanf("%d", &nodes[i]); buildTree(nodes);
}
```

1.60 Wavelet Tree

```
const int MAX = 100005;
// if array elements are small (<=1e6), replace rm[*it] with *it.
int ara[MAX]; set <int> S; map <int, int> M; int rm[MAX];
struct wavelet_tree {
int lo, hi; wavelet_tree *l, *r;
vector <int> b; vector <ll> c;
wavelet_tree(int *from, int *to, int x, int y) {
  lo = x, hi = y; if( from >= to) return;
  if( hi == lo ) {
    b.reserve(to - from + 1); b.push_back(0);
    c.reserve(to - from + 1); c.push_back(0);
    for(auto it = from; it != to; it++) {
```

```
      b.push_back(b.back() + 1);
      c.push_back(c.back() + rm[*it]);
    } return; }
  int mid = (lo+hi)/2;
  auto f = [mid](int x) { return x <= mid; };
  b.reserve(to - from + 1); b.push_back(0);
  c.reserve(to - from + 1); c.push_back(0);
  for(auto it = from; it != to; it++) {
    b.push_back(b.back() + f(*it));
    c.push_back(c.back() + rm[*it]); }
  auto pivot = stable_partition(from, to, f);
  l = new wavelet_tree(from, pivot, lo, mid);
  r = new wavelet_tree(pivot, to, mid + 1, hi);
}
// k'th smallest element in subarray [l,r]
int kth(int l, int r, int k) {
  if(l > r) return 0; if(lo == hi) return lo;
  int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r];
  if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
  return this->r->kth(l - lb, r - rb, k - inLeft); }
// number of elements <= k in subarray [l,r]
int LTE(int l, int r, int k) {
  if(l > r or k < lo) return 0; if(hi <= k) return r - l + 1;
  int lb = b[l-1], rb = b[r];
  return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k); }
// number of occurrences of k in subarray [l,r]
int count(int l, int r, int k) {
  if(l > r or k < lo or k > hi) return 0;
  if(lo == hi) return r - l + 1;
  int lb = b[l - 1], rb = b[r], mid = (lo + hi)/2;
  if(k <= mid) return this->l->count(lb + 1, rb, k);
  return this->r->count(l - lb, r - rb, k); }
// sum of the elements <= k in subarray [l,r]
ll sumk(int l, int r, int k) {
  if(l > r or k < lo) return 0;
  if(hi <= k) return c[r] - c[l - 1];
  int lb = b[l - 1], rb = b[r];
  return this->l->sumk(lb + 1, rb, k) + this->r->sumk(l - lb, r - rb, k); }
~wavelet_tree() { delete l; delete r; }
};
int main() { int n; cin >> n;
    for(int i=1;i<=n;i++) cin >> ara[i], S.insert(ara[i]);
    int it = 0; for(int e : S) M[e] = ++it, rm[it] = e;
    for(int i=1;i<=n;i++) ara[i] = M[ara[i]];
    wavelet_tree *Tree = new wavelet_tree(ara + 1, ara + n + 1, 1, S.size());
    /***
    to know the sum of elements in range [a, b] which are <= c
    auto it = S.upper_bound(c);
    if(it == S.begin()) sum = 0
    else it--, sum = Tree->sumk(a, b, M[*it]);
    ***/
}
```

2 Notes

3 Notes

3.1 Geometry

3.1.1 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: $\dfrac{a+b}{a-b} = \dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

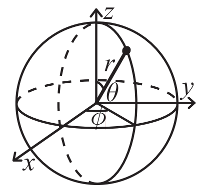
3.1.2 Quadrilaterals

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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

3.1.3 Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \operatorname{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \operatorname{atan2}(y,x)$$

3.2 Sums

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

3.3 Series

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

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

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

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

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

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

3.4 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.

3.5 Number Theory

3.5.1 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

3.5.2 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 $n < 1e19$.

3.5.3 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.

3.5.4 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 $n-1$.

3.5.5 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 $\sum_{d|n}\mu(d)f(d) = \prod_{p|n}(1-f(p))$, $\sum_{d|n}\mu(d)^2 f(d) = \prod_{p|n}(1+f(p))$.

3.5.6 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: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod p$.

3.5.7 Jacobi symbol

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

3.5.8 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 $nu \equiv i \pmod p$.)

3.5.9 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.

3.5.10 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 $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

3.5.11 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 $(a-1)(b-1) - 1 = ab - a - b$.

3.5.12 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.

3.6 Permutations

3.6.1 Derangements

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

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

3.6.2 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

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

3.7 Partitions and subsets

3.7.1 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

3.8 General purpose numbers

3.8.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

3.8.2 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. $\pi(j) > j$.

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

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

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

3.8.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

3.8.4 Bell numbers

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

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

3.8.5 Bernoulli numbers

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

3.8.6 Catalan numbers

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

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

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

- 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.

3.9 Games

3.9.1 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 $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

3.9.2 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.

3.9.3 Misère Nim

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

3.10 Optimization tricks

3.10.1 Bit hacks

- `x & -x` is the least bit in `x`
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.

3.10.2 Pragmas

- #pragma GCC optimize ("Ofast") will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- #pragma GCC target ("avx,avx2") can double performance of vectorized code, but causes crashes on old machines.
- #pragma GCC optimize ("trapv") kills the program on integer overflows (but is really slow).