BIT 1D

```
template<class type=int>
struct fenwickTree {
     int n,logn;    type T[maxn]={};
     fenwickTree(int nn){    n=1; logn=0;   while (n<nn) n*=2,logn++;    }
     void update(int id,type val){     for (int i=id; i<=n;i+=i&-i) T[i]+=val;     }
     type query(int id){
          type r=0; for (int i=id;i>=1;i-=i&-i)  r+=T[i];
          return r;
     }
     type query(int ll,int rr){   return query(rr)-query(ll-1);     }
     int lower_bound(int r,int l,type val){
          for(int i=l-1;i>=0;i--)
               if(T[r-(1<<i)]>=val)    return lower_bound(r-(1<<i),i,val);
               else val-=T[i];
          return r;
     }
     int lower_bound(type val){
          if (val>T[n]) return 0; else return lower_bound(n,logn,val);
     }
};
```

BIT 2D

```
template<class type=int>
struct fenwickTree {
     int m,n;  type T[maxn][maxn]={};
     fenwickTree(int mm,int nn){  m=mm; n=nn;    }
     void init(int mm,int nn){    m=mm; n=nn;    memset(T,0,sizeof(T));  }
     void update(int p,int q,type val){
          for (int i=p;i<=m;i+=i&-i)   for (int j=q;j<=n;j+=j&-j)   T[i][j]+=val;
     }
     type query(int p,int q){
          type r=0;
          for (int i=p;i>=1;i-=i&-i)   for (int j=q;j>=1;j-=j&-j)   r+=T[i][j];
          return r;
     }
     type query(int p,int q,int pp,int qq){
          return query(pp,qq)-query(pp,q-1)-query(p-1,qq)+query(p-1,q-1);
     }
};
```

Heavy light decomposition

```
const int MN = 100111;  vector<int> G[MN];
int sz[MN], dep[MN], fa[MN], dfn[MN], top[MN], n, id;
int querySeg(int i, int l, int r, int u, int v) {return 0;}
void dfs1(int u, int f = 0) {
    sz[u] = 1; fa[u] = f;
    for (auto &v:G[u]) if (v!=f){ dep[v]=dep[u]+1; dfs1(v,u); sz[u]+=sz[v];  }
}
void dfs2(int u, int chain, int f = 0) {
    int son(-1); dfn[u] = ++id; top[u] = chain;
    for(auto &v:G[u]) if(v!=f)    if(son==-1||sz[son]<sz[v])   son=v;
    if (~son) dfs2(son, chain, u);
    for (auto &v: G[u]) if (v != f && v != son) dfs2(v, v, u);
}
int query(int u, int v) {
    int res = 0, fu = top[u], fv = top[v];
    while (fu != fv) {
        if (dep[fu] < dep[fv]) swap(u, v), swap(fu, fv);
        res+=querySeg(1,1,n,dfn[fu],dfn[u]); u=fa[fu],fu=top[u];
    }
    if (dep[u] > dep[v]) swap(u, v);
    res += querySeg(1, 1, n, dfn[u], dfn[v]);
    return res;
}
int main() {  dfs1(1);  id = 0;   dfs2(1, 1);    }
```

LCA (Sparse table)

```
vector<int> a[maxn];    int n,m,u,v,root,L[maxn],P[21][maxn];
int level(int u){
     if (u==root) return L[u]=1;
     if (L[u]) return L[u];
     return L[u]=level(P[0][u])+1;
}
void initLCA(){
     for (int i=1;i<=n;i++)  if (P[0][i]==0) root=i;
     for (int i=1;i<=n;i++)  level(i);
     for (int k=1;k<=20;k++) for (int i=1;i<=n;i++)  P[k][i]=P[k-1][P[k-1][i]];
}
int getLCA(int u,int v){
     for (int k=20;k>=0;k--) if (L[P[k][u]]>=L[v])   u=P[k][u];
     for (int k=20;k>=0;k--) if (L[P[k][v]]>=L[u])   v=P[k][v];
     for (int k=20;k>=0;k--) if (P[k][u]!=P[k][v])   u=P[k][u],v=P[k][v];
     while (u!=v)  u=P[0][u],v=P[0][v];
     return u;
}
void resetLCA(){
     memset(L,0,sizeof(L)); for (int i=1;i<maxn;i++){ a[i].clear(); P[0][i]=0; }
}
```

RMQ

```
int n,a[maxn], log_2[maxn], rmq[maxl][maxn];
void initlog_2(){  log_2[1]=0;   for (int i=2;i<=n;i++)  log_2[i]=log_2[i/2]+1;  }
void initRMQ(){
     for (int i=1;i<=n;i++)  rmq[0][i]=a[i];
     for (int k=1;(1<<k)<=n;k++)  for (int i=1;i+(1<<k)-1<=n;i++)
          rmq[k][i]=min(rmq[k-1][i],rmq[k-1][i+(1<<(k-1))]);
}
int getRMQ(int u,int v){
     int k=log_2[v-u+1];return min(rmq[k][u],rmq[k][v-(1<<k)+1]);
}
```

Segment tree 1D

```
template<class type=int>
struct segmentTree{
     struct nodetree{
          type tmax,tmin,lazy; int imax,imin;   bool isErase;
          nodetree(){ tmax=-oo; tmin=oo; imax=-1; imin=-1; lazy=0; isErase=false; }
     };
```

```
nodetree a[4*maxn]; int imax,imin; type tmax,tmin;
void merge(int node){
     if (a[node*2].tmax>=a[node*2+1].tmax){
          a[node].tmax=a[node*2].tmax; a[node].imax=a[node*2].imax;
     } else {
          a[node].tmax=a[node*2+1].tmax; a[node].imax=a[node*2+1].imax;
     }
     if (a[node*2].tmin<=a[node*2+1].tmin){
          a[node].tmin=a[node*2].tmin; a[node].imin=a[node*2].imin;
     } else {
          a[node].tmin=a[node*2+1].tmin; a[node].imin=a[node*2+1].imin;
     }
}
void build(int node,int ll,int rr,int u,int v){
     if (v<ll || rr<u || ll>rr) return;
     if (u<=ll && ll==rr && rr<=v){
          a[node].tmax=arr[ll]; a[node].tmin=arr[ll];
          a[node].imax=ll; a[node].imin=ll;
     } else {
          build(node*2,ll,(ll+rr)/2,u,v); build(node*2+1,(ll+rr)/2+1,rr,u,v);
          merge(node);
     }
}
void diffuse(int node,int ll,int rr){
     if (a[node].lazy==0) return;
     if (a[node].lazy==-1){
          a[node].tmax=-oo; a[node].tmin=oo;
          if (ll!=rr){ a[node*2].lazy=-1; a[node*2+1].lazy=-1; }
     } else {
          a[node].tmax+=a[node].lazy; a[node].tmin+=a[node].lazy;
          if (ll!=rr){
               a[node*2].lazy+=a[node].lazy; a[node*2+1].lazy+=a[node].lazy;
          }
          a[node].lazy=0;
     }
}
void update(int node,int ll,int rr,int u,int v,type val){
     diffuse(node,ll,rr); if (v<ll || rr<u || ll>rr) return;
     if (u<=ll && ll<=rr && rr<=v){
          if (val==0)   a[node].lazy=-1;   else a[node].lazy+=val;
          diffuse(node,ll,rr);
     } else {
          update(node*2,ll,(ll+rr)/2,u,v,val);
          update(node*2+1,(ll+rr)/2+1,rr,u,v,val);
          merge(node);
     }
}
void init(){ tmax=-oo; tmin=oo; imax=-1; imin=-1; }
void query(int node,int ll,int rr,int u,int v){
     diffuse(node,ll,rr); if (v<ll || rr<u || ll>rr) return;
     if (u<=ll && ll<=rr && rr<=v){
          if (tmax<a[node].tmax){ tmax=a[node].tmax; imax=a[node].imax; }
          if (tmin>a[node].tmin){ tmin=a[node].tmin; imin=a[node].imin; }
     } else {
          query(node*2,ll,(ll+rr)/2,u,v); query(node*2+1,(ll+rr)/2+1,rr,u,v);
     }
}
};
```

Segment tree 2D

```
struct node {
     int id,ll,rr;
     node(int i,int l,int r){     id=i; ll=l; rr=r;  }
     node left() const {return node(id*2,ll,(ll+rr)/2);  }
     node right() const {    return node(id*2+1,(ll+rr)/2+1,rr);   }
     bool irrelevant(int l,int r) const {  return ll>r || l>rr || l>r;  }
};
template<class type=int>
struct segmentTree {
     type T[4*maxn][4*maxn]; int size;
     void init(int m,int n){ size=n*2; memset(T,0,sizeof(T));  }
     void update(const node &dx, const node &dy,int u,int v,type val, bool all){
          if (dx.irrelevant(u,u) || dy.irrelevant(v,v))   return;
          if (dx.ll==dx.rr && dy.ll==dy.rr){ T[dx.id][dy.id]+=val; return; }
          if (all){
               if (dx.ll!=dx.rr){
                    update(dx.left(),dy,u,v,val,true);
                    update(dx.right(),dy,u,v,val,true);
                    for (int i=1;i<size;i++)
                         T[dx.id][i]=T[dx.left().id][i]+T[dx.right().id][i];
               } else {  update(dx,dy,u,v,val,false); }
          } else {
               if (dy.ll!=dy.rr){
                    update(dx,dy.left(),u,v,val,false);
                    update(dx,dy.right(),u,v,val,false);
                    T[dx.id][dy.id]=T[dx.id][dy.left().id]+T[dx.id][dy.right().id];
               } else { T[dx.id][dy.id]+=val; }
          }
     }
     type query(const node &dx, const node &dy,int u,int v,int uu,int vv){
          if (dx.irrelevant(u,uu) || dy.irrelevant(v,vv)) return 0;
          if (u<=dx.ll && dx.rr<=uu){
               if (v<=dy.ll && dy.rr<=vv) return T[dx.id][dy.id];
               type sum1=query(dx,dy.left(),u,v,uu,vv);
               type sum2=query(dx,dy.right(),u,v,uu,vv);
               return sum1+sum2;
          } else {
               type sum1=query(dx.left(),dy,u,v,uu,vv);
               type sum2=query(dx.right(),dy,u,v,uu,vv);
               return sum1+sum2;
          }
     }
};
segmentTree<int> it; node dx_(1,1,maxn*2); node dy_(1,1,maxn*2);
```

2D (Point)

```
#define EPS 1e-6
```

```
const double PI = acos(-1.0);
double DEG_to_RAD(double d) { return d*PI/180.0; }
double RAD_to_DEG(double r) { return r*180.0/PI; }
inline int cmp(double a, double b) {
     return (a < b-EPS) ? -1 : ((a > b+EPS) ? 1 : 0);
}
struct Point {
     double x,y;    Point() { x = y = 0.0; }
     Point(double x,double y) : x(x),y(y) {}
     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 * (double k) const { return Point(x*k, y*k); }
     Point operator / (double k) const { return Point(x/k, y/k); }
     double operator * (const Point& a) const { return x*a.x + y*a.y; } // dot product
     double operator % (const Point& a) const { return x*a.y - y*a.x; } // cross product
     int cmp(Point q) const {
          if (int t = ::cmp(x,q.x)) return t;
          return ::cmp(y,q.y);
     }
#define Comp(x) bool operator x (Point q) const { return cmp(q) x 0; }
     Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=) Comp(!=)
#undef Comp
     Point conj() { return Point(x, -y); }
     double norm() { return x*x + y*y; }
     double len() { return sqrt(norm()); }
     Point rotate(double alpha) {
          double cosa = cos(alpha), sina = sin(alpha);
          return Point(x * cosa - y * sina, x * sina + y * cosa);
     }
};
int ccw(Point a, Point b, Point c) {  return cmp((b-a)%(c-a),0);   }
int RE_TRAI = ccw(Point(0, 0), Point(0, 1), Point(-1, 1));
int RE_PHAI = ccw(Point(0, 0), Point(0, 1), Point(1, 1));
istream& operator>>(istream& cin,Point& p){ cin >> p.x >> p.y; return cin; }
ostream& operator<<(ostream& cout,Point& p){ cout << p.x << ' ' << p.y; return cout; }
double angle(Point a, Point o, Point b) {
     a = a - o; b = b - o; return acos((a * b) / sqrt(a.norm()) / sqrt(b.norm()));
}
double directed_angle(Point a, Point o, Point b) {
     double t = -atan2(a.y - o.y, a.x - o.x)+atan2(b.y - o.y, b.x - o.x);
     while (t < 0) t += 2*PI;     return t;
}
double distToLine(Point p, Point a, Point b, Point &c) {
     Point ap = p - a, ab = b - a; double u = (ap * ab) / ab.norm();
     c = a + (ab * u); return (p-c).len();
}
double distToLineSegment(Point p, Point a, Point b, Point &c) {
     Point ap = p - a, ab = b - a; double u = (ap * ab) / ab.norm();
     if (u < 0.0) { c = Point(a.x, a.y); return (p - a).len(); }
     if (u > 1.0) { c = Point(b.x, b.y); return (p - b).len(); }
     return distToLine(p, a, b, c);
}
```

2D (Line)

```
struct Line {
     double a, b, c; Point A, B; // Added for polygon intersect line.
     Line(double a, double b, double c) : a(a), b(b), c(c) {}
     Line(Point A,Point B):A(A),B(B){ a=B.y-A.y; b=A.x-B.x; c=-(a*A.x+b*A.y); }
     Line(Point P, double m) { a = -m; b = 1; c = -((a * P.x) + (b * P.y)); }
     double f(Point A) { return a*A.x + b*A.y + c; }
};
bool areParallel(Line l1, Line l2) { return cmp(l1.a*l2.b, l1.b*l2.a) == 0; }
bool areSame(Line l1, Line l2) {
     return
areParallel(l1,l2)&&cmp(l1.c*l2.a,l2.c*l1.a)==0&&cmp(l1.c*l2.b,l1.b*l2.c)==0;
}
bool areIntersect(Line l1, Line l2, Point &p) {
     if (areParallel(l1, l2)) return false;
     double dx = l1.b*l2.c - l2.b*l1.c; double dy = l1.c*l2.a - l2.c*l1.a;
     double d  = l1.a*l2.b - l2.a*l1.b; p = Point(dx/d, dy/d); return true;
}
void closestPoint(Line l, Point p, Point &ans) {
     if (fabs(l.b) < EPS) { ans.x = -(l.c) / l.a; ans.y = p.y; return; }
     if (fabs(l.a) < EPS) { ans.x = p.x; ans.y = -(l.c) / l.b; return; }
     Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y)); areIntersect(l, perp, ans);
}
void reflectionPoint(Line l, Point p, Point &ans) {
     Point b; closestPoint(l, p, b); ans = p + (b - p) * 2;
}
```

2D (Circle)

```
struct Circle : Point {
     double r;
     Circle(double x = 0, double y = 0, double r = 0) : Point(x, y), r(r) {}
     Circle(Point p, double r) : Point(p), r(r) {}
     bool contains(Point p) { return (*this - p).len() <= r + EPS; }
};
double sqr(double a){   return a*a;    }
void tangents(Point c, double r1, double r2, vector<Line> & ans) {
     double r = r2 - r1; double z = sqr(c.x) + sqr(c.y); double d = z - sqr(r);
     if (d < -EPS)  return;
     d=sqrt(fabs(d)); Line l((c.x*r+c.y*d)/z,(c.y*r-c.x*d)/z,r1);
     ans.push_back(l);
}
vector<Line> tangents(Circle a, Circle b) {
     vector<Line> ans; ans.clear();
     for (int i=-1; i<=1; i+=2) for (int j=-1; j<=1; j+=2)
          tangents(b-a, a.r*i, b.r*j, ans);
     for (int i = 0; i < ans.size(); ++i)
          ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
     vector<Line> ret;
     for (int i = 0; i < ans.size(); ++i) {
          bool ok = true;
          for (int j = 0; j < ret.size(); ++j)
               if (areSame(ret[j], ans[i])) { ok = false; break; }
          if (ok) ret.push_back(ans[i]);
     }
     return ret;
```

```
}
vector<Point> intersection(Line l, Circle cir) {
     double r=cir.r,a=l.a,b=l.b,c=l.c+l.a*cir.x+l.b*cir.y; vector<Point> res;
     double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
     if (c*c > r*r*(a*a+b*b)+EPS) return res;
     else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
          res.push_back(Point(x0, y0) + Point(cir.x, cir.y)); return res;
     } else {
          double d = r*r - c*c/(a*a+b*b); double mult = sqrt (d / (a*a+b*b));
          double ax,ay,bx,by; ax = x0 + b * mult;
          bx = x0 - b * mult; ay = y0 - a * mult; by = y0 + a * mult;
          res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
          res.push_back(Point(bx, by) + Point(cir.x, cir.y));
          return res;
     }
}
double cir_area_solve(double a,double b,double c){ return acos((a*a+b*b-c*c)/2/a/b); }
double cir_area_cut(double a, double r) {
     double s1 = a * r * r / 2; double s2 = sin(a) * r * r / 2; return s1 - s2;
}
double commonCircleArea(Circle c1, Circle c2) {
     if (c1.r < c2.r) swap(c1, c2);
     double d = (c1 - c2).len(); if (d + c2.r <= c1.r + EPS) return c2.r*c2.r*PI;
     if (d >= c1.r + c2.r - EPS) return 0.0;
     double a1 = cir_area_solve(d,c1.r,c2.r); double a2 = cir_area_solve(d,c2.r,c1.r);
     return cir_area_cut(a1*2, c1.r) + cir_area_cut(a2*2, c2.r);
}
bool areIntersect(Circle u, Circle v) {
     if (cmp((u - v).len(), u.r + v.r) > 0) return false;
     if (cmp((u - v).len() + v.r, u.r) < 0) return false;
     if (cmp((u - v).len() + u.r, v.r) < 0) return false;
     return true;
}
vector<Point> circleIntersect(Circle u, Circle v) {
     vector<Point> res; if (!areIntersect(u, v)) return res;
     double d=(u-v).len(); double alpha=acos((u.r*u.r+d*d-v.r*v.r)/2.0/u.r/d);
     Point p1=(v-u).rotate(alpha); Point p2=(v-u).rotate(-alpha);
     res.push_back(p1/p1.len()*u.r+u); res.push_back(p2/p2.len()*u.r+u);
     return res;
}
```

2D (Polygon)

```
typedef vector<Point> Polygon;
double area2(Point a, Point b, Point c) { return a%b + b%c + c%a; }
#ifdef REMOVE_REDUNDANT
bool between(const Point &a, const Point &b, const Point &c) {
     return (fabs(area2(a,b,c))<EPS&&(a.x-b.x)*(c.x-b.x)<=0&&(a.y-b.y)*(c.y-b.y)<=0);
}
#endif
void ConvexHull(Polygon &pts) {
     sort(pts.begin(),pts.end()); pts.erase(unique(pts.begin(),pts.end()),pts.end());
     Polygon up, dn;
     for (int i = 0; i < pts.size(); i++) {
          while              (up.size()>1&&area2(up[up.size()-2],up.back(),pts[i])>=0)
up.pop_back();
          while              (dn.size()>1&&area2(dn[dn.size()-2],dn.back(),pts[i])<=0)
dn.pop_back();
          up.push_back(pts[i]); dn.push_back(pts[i]);
     }
     pts=dn; for(int i=(int)up.size()-2;i>=1;i--) pts.push_back(up[i]);
#ifdef REMOVE_REDUNDANT
     if (pts.size() <= 2) return;
     dn.clear(); dn.push_back(pts[0]); dn.push_back(pts[1]);
     for (int i = 2; i < pts.size(); i++) {
          if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
          dn.push_back(pts[i]);
     }
     if        (dn.size()>=3&&between(dn.back(),dn[0],dn[1])){        dn[0]=dn.back();
dn.pop_back(); }
     pts = dn;
#endif
}
Polygon convex_hull(Polygon P){ Polygon tmp=P; ConvexHull(tmp); return tmp; }
double signed_area(Polygon 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 area(const Polygon &p){ return fabs(signed_area(p)); }
Point centroid(Polygon p) {
     Point c(0,0); double scale = 6.0 * signed_area(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;
}
double perimeter(Polygon P) {
     double res = 0;
     for(int i = 0; i < P.size(); ++i) {
          int j = (i+1) % P.size(); res += (P[i] - P[j]).len();
     }
     return res;
}
bool is_convex(const Polygon &P) {
     int sz = (int) P.size(); if (sz <= 2) return false;
     int isLeft = ccw(P[0], P[1], P[2]);
     for (int i = 1; i < sz; i++)
          if(ccw(P[i],P[(i+1)%sz],P[(i+2)%sz])*isLeft<0) return false;
     return true;
}
bool in_polygon(const Polygon &p, Point q) {
     if ((int)p.size() == 0) return false;
     int n = (int) p.size();
     for (int i = 0; i < n; i++) {
          int j=(i+1)%n; Point u=p[i],v=p[j]; if(u>v)     swap(u,v);
```

```
          if(ccw(u,v,q)==0&&u<=q&&q<=v)     return true;
     }
     int c = 0;
     for (int i = 0; i < n; i++) {
          int j = (i + 1) % n;
          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;
}
#define Det(a,b,c) \
     ((double)(b.x-a.x)*(double)(c.y-a.y) \
     -(double)(b.y-a.y)*(double)(c.x-a.x))
bool in_convex(Polygon& l, Point p) {
     int a=1,b=l.size()-1,c; if(Det(l[0],l[a],l[b])>0)    swap(a,b);
     if (Det(l[0], l[a], p) >= 0 || Det(l[0], l[b], p) <= 0) return false;
     while (abs(a-b) > 1) {
          c=(a+b)/2;    if(Det(l[0],l[c],p)>0) b=c;  else a=c;
     }
     return Det(l[a], l[b], p) < 0;
}
Polygon polygon_cut(const Polygon& P, Line l) {
     Polygon Q;
     for (int i = 0; i < P.size(); ++i) {
          Point A = P[i], B = (i == P.size()-1) ? P[0] : P[i+1];
          if (ccw(l.A, l.B, A) != -1) Q.push_back(A);
          if (ccw(l.A, l.B, A)*ccw(l.A, l.B, B) < 0) {
               Point p; areIntersect(Line(A, B), l, p); Q.push_back(p);
          }
     }
     return Q;
}
bool intersect_1pt(Point a, Point b, Point c, Point d, Point &r) {
     double D =  (b - a) % (d - c); if (cmp(D, 0) == 0) return false;
     double t=((c-a)%(d-c))/D; double s=-((a-c)%(b-a))/D; r=a+(b-a)*t;
     return cmp(t,0)>=0&&cmp(t,1)<=0&&cmp(s,0)>=0&&cmp(s,1)<=0;
}
Polygon convex_intersect(Polygon P, Polygon Q) {
     const int n=P.size(),m=Q.size(); int a=0,b=0,aa=0,ba=0;
     enum { Pin, Qin, Unknown } in = Unknown; Polygon R;
     do {
          int a1=(a+n-1)%n,b1=(b+m-1)%m; double C=(P[a]-P[a1])%(Q[b]-Q[b1]); Point r;
          double A=(P[a1]-Q[b])%(P[a]-Q[b]); double B=(Q[b1]-P[a])%(Q[b]-P[a]);
          if (intersect_1pt(P[a1], P[a], Q[b1], Q[b], r)) {
               if(in == Unknown)  aa = ba = 0;
               R.push_back( r ); in = B > 0 ? Pin : A > 0 ? Qin : in;
          }
          if (C == 0 && B == 0 && A == 0) {
               if(in==Pin){ b=(b+1)%m; ++ba; } else{ a=(a+1)%m; ++aa; }
          } else if (C >= 0) {
               if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
               else       { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
          } else {
               if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
               else       { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
          }
     } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
     if (in == Unknown) {
          if (in_convex(Q, P[0])) return P;
          if (in_convex(P, Q[0])) return Q;
     }
     return R;
}
double convex_diameter(Polygon pt) {
     const int n = pt.size(); int is = 0, js = 0;
     for (int i = 1; i < n; ++i) {
          if (pt[i].y > pt[is].y) is = i;
          if (pt[i].y < pt[js].y) js = i;
     }
     double maxd=(pt[is]-pt[js]).norm(); int i,maxi,j,maxj; i=maxi=is; j=maxj=js;
     do {
          int jj=j+1; if(jj==n) jj=0;
          if((pt[i]-pt[jj]).norm()>(pt[i]-pt[j]).norm())  j=(j+1)%n; else i=(i+1)%n;
          if((pt[i]-pt[j]).norm()>maxd){ maxd=(pt[i]-pt[j]).norm(); maxi=i; maxj=j; }
     } while (i != is || j != js);
     return maxd; /* farthest pair is (maxi, maxj). */
}
#define MAXN 100
double mindist = 1e20;  Point x,y;
bool cmpy(Point u, Point v) {
     if (u.x==v.x) return u.y<v.y;
     return u.x<v.x;
}
void upd_ans(Point _x, Point _y){ x=_x; y=_y;    }
void rec(int l, int r, Point a[]) {
     if (r - l <= 3) {
          for (int i=l; i<=r; ++i) for (int j=i+1; j<=r; ++j) upd_ans(a[i], a[j]);
          sort(a+l, a+r+1, cmpy); return;
     }
     int m=(l+r)>>1; int midx=a[m].x; rec(l,m,a), rec(m+1,r,a);
     static Point t[MAXN]; merge(a+l,a+m+1,a+m+1,a+r+1,t,cmpy); copy(t,t+r-l+1,a+l);
     int tsz = 0;
     for (int i=l; i<=r; ++i)
     if (fabs(a[i].x - midx) < mindist) {
          for(int j=tsz-1;j>=0&&a[i].y-t[j].y<mindist;--j) upd_ans(a[i],t[j]);
          t[tsz++]=a[i];
     }
}
bool isSquare(long long x) { long long tmp=(long long)sqrt(x); return (x==tmp*tmp); }
bool isIntegerCoordinates(int x, int y, int z) {
     long long s=(long long)(x+y+z)*(x+y-z)*(x+z-y)*(y+z-x);
     return (s%4==0 && isSquare(s/4));
}
```

Convex hull (Monotone chain)

```
struct Point {
     int X,Y;
```

```
     bool operator<(const Point &v){ return X==v.X?Y<v.Y:X<v.X; }
     int cross(const Point &p, const Point &q) const {
          return (p.X-X)*(q.Y-Y)-(p.Y-Y)*(q.X-X);
     }
};
int n; Point a[maxn],p[maxn];
void monotonechain(){
     sort(a+1,a+n+1); int k=1;
     for (int i=1;i<=n;i++){
          while (k>=3 && p[k-2].cross(p[k-1],a[i])<=0) k--;
          p[k++]=a[i];
     }
     for (int i=n-1,t=k+1;i>0;i--){
          while (k>=t && p[k-2].cross(p[k-1],a[i])<=0) k--;
          p[k++]=a[i];
     }
     for (int i=1;i<k;i++)   a[i]=p[i];
     n=k-2;
}
```

Convex hull (Graham)

```
Point O; int n; Point a[maxn];
void operator -= (Point &a,Point b){ a.X-=b.X; a.Y-=b.Y; }
bool ccw(Point o,Point a,Point b){ a-=o; b-=o; return a.X*b.Y>a.Y*b.X; }
bool cmp(Point a,Point b){ return ccw(O,a,b); }
void graham(){
     sort(a+1,a+n+1); O=a[1]; sort(a+2,a+n+1,cmp); a[0]=a[n]; a[n+1]=a[1]; int j=1;
     for (int i=1;i<=n+1;i++){
          while (j>2 && !ccw(a[j-2],a[j-1],a[i])) j--;
          a[j++]=a[i];
     }
     n=j-2;
}
```

Trick

```
int pointer;  vector<long long> M,B;
bool bad(int l1,int l2,int l3){
     return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
void add(long long m,long long b){
     M.push_back(m);    B.push_back(b);
     while (M.size()>=3 && bad(M.size()-3,M.size()-2,M.size()-1)){
          M.erase(M.end()-2);B.erase(B.end()-2);
     }
}
long long query(long long x){
     if (pointer >=M.size()) pointer=M.size()-1;
     while(pointer<M.size()-1&&M[pointer+1]*x+B[pointer+1]>M[pointer]*x+B[pointer])
          pointer++;
     return M[pointer]*x+B[pointer];
}
```

Smallest enclosing circle

```
struct SmallestEnclosingCircle {
     Circle getCircumcircle(Point a, Point b, Point c) {
          double d=2.0*(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y));
          assert(fabs(d)>EPS);
          double x = (a.norm()*(b.y-c.y)+b.norm()*(c.y-a.y)+c.norm()*(a.y-b.y))/d;
          double y = (a.norm()*(c.x-b.x)+b.norm()*(a.x-c.x)+c.norm()*(b.x-a.x))/d;
          Point p(x, y); return Circle(p, (p-a).len());
     }
     Circle getCircle(vector<Point> points) {
          assert(!points.empty()); random_shuffle(points.begin(), points.end());
          Circle c(points[0], 0); int n = points.size();
          for (int i = 1; i < n; i++) if ((points[i]-c).len() > c.r+EPS) {
               c = Circle(points[i], 0);
               for(int j=0;j<i;j++) if((points[j]-c).len()>c.r+EPS){
                    c=Circle((points[i]+points[j])/2,(points[i]-points[j]).len()/2);
                    for(int k=0;k<j;k++) if((points[k]-c).len()>c.r+EPS)
                         c=getCircumcircle(points[i],points[j],points[k]);
               }
          }
          return c;
     }
};
```

Matching un-weight

```
const int maxn = 1010, maxm = 50010;
struct match_0{
     int n,m,start,finish,newroot,qsize,adj[maxm],next[maxm],last[maxn],
          mat[maxn],que[maxn],dad[maxn],root[maxn];
     bool inque[maxn],inpath[maxn],inblossom[maxn];
     void init(int _n){
          n=_n; m=0; for (int x=1;x<=n;x++){ last[x]=-1; mat[x]=0; }
     }
     void add(int u,int v){ adj[m]=v; next[m]=last[u]; last[u]=m++; }
     int lca(int u,int v){
          for (int x=1;x<=n;x++) inpath[x]=0;
          while (true){
               u=root[u]; inpath[u]=1; if (u==start) break;
               u=dad[mat[u]];
          }
          while (true){
               v=root[v]; if (inpath[v]) break;
               v=dad[mat[v]];
          }
          return v;
     }
     void trace(int u){
          while (root[u]!=newroot){
               int v=mat[u]; inblossom[root[u]]=1; inblossom[root[v]]=1;
               u=dad[v]; if (root[u]!=newroot) dad[u]=v;
          }
     }
     void blossom(int u,int v){
          for (int x=1;x<=n;x++) inblossom[x]=0;
          newroot=lca(u,v); trace(u); trace(v);
          if (root[u]!=newroot) dad[u]=v;
          if (root[v]!=newroot) dad[v]=u;
```

```
          for (int x=1;x<=n;x++){
               if (inblossom[root[x]]){
                    root[x]=newroot;
                    if (!inque[x]){ inque[x]=1; que[qsize++]=x; }
               }
          }
     }
     bool bfs(){
          for (int x=1;x<=n;x++){ inque[x]=0; dad[x]=0; root[x]=x; }
          qsize=0; que[qsize++]=start; inque[start]=1; finish=0;
          for (int i=0,u;i<qsize;i++){
               u=que[i];
               for (int e=last[u],v;e!=-1;e=next[e]){
                    v=adj[e];
                    if (root[v]!=root[u] && v!=mat[u]){
                         if (v==start||(mat[v]>0&&dad[mat[v]]>0)) blossom(u,v);
                         else if (dad[v]==0){
                              dad[v]=u;
                              if (mat[v]>0) que[qsize++]=mat[v];
                              else { finish=v; return true; }
                         }
                    }
               }
          }
          return false;
     }
     void enlarge(){
          int u=finish,v,x;
          while(u>0){ v=dad[u]; x=mat[v]; mat[v]=u; mat[u]=v; u=x; }
     }
     int maxMatching(){
          for (int x=1;x<=n;x++)
               if (mat[x]==0){ start=x; if(bfs()) enlarge(); }
          int total=0; for(int x=1;x<=n;x++) if(mat[x]>x) total++;
          return total;
     }
}match0;
void solve_match0(){
     int n,m,u,v,k; cin>>n>>m; match0.init(n+m);
     while(cin>>u>>v){ v+=n; match0.add(u,v); match0.add(v,u); }
     cout<<match0.maxMatching()<<"\n";
     for(int x=1,y;x<=n;x++){ y=match0.mat[x]; if(y!=0) cout<<x<<" "<<y-n<<"\n"; }
}
```

Matching weight

```
struct match_2{
     int n; VI d,arg,trace,fx,fy,mx,my; VVI cost,adj; queue<int> qu;
     void init(int _nx,int _ny){
          n=max(_nx,_ny); cost=VVI(n,VI(n,oo)); adj=VVI(n); d=VI(n); arg=VI(n);
          trace=VI(n); fx=VI(n); fy=VI(n); mx=VI(n,-1); my=VI(n,-1);
     }
     void add(int u,int v,int w){
          if (cost[u][v]==oo) adj[u].push_back(v);
          if (cost[u][v]>w) cost[u][v]=w;
     }
     int getCost(int x,int y){ return cost[x][y]-fx[x]-fy[y]; }
     void initBFS(int start){
          for (int i=0;i<n;i++) trace[i]=-1;
          while (qu.size()) qu.pop();
          qu.push(start);
          for (int i=0;i<n;i++){ d[i]=getCost(start,i); arg[i]=start; }
     }
     int findPath(){
          int x,y,w;
          while (qu.size()){
               x=qu.front(); qu.pop();
               for (int i=0;i<adj[x].size();i++){
                    y=adj[x][i];
                    if (trace[y]==-1){
                         w=getCost(x,y);
                         if (w==0){
                              trace[y]=x; if (my[y]==-1) return y;
                              qu.push(my[y]);
                         }
                         if (d[y]>w){ d[y]=w; arg[y]=x; }
                    }
               }
          }
          return -1;
     }
     int update(int start){
          int delta=oo;
          for (int y=0;y<n;y++) if (trace[y]==-1) delta=min(delta,d[y]);
          fx[start]+=delta;
          for (int y=0,x;y<n;y++)
               if (trace[y]!=-1){ x=my[y]; fx[x]+=delta; fy[y]-=delta; }
               else d[y]-=delta;
          for (int y=0;y<n;y++)
               if (trace[y]==-1 && d[y]==0){
                    trace[y]=arg[y]; if (my[y]==-1) return y;
                    qu.push(my[y]);
               }
          return -1;
     }
     void enlarge(int finish){
          for (int y=finish,x,yy;y!=-1; ){
               x=trace[y]; yy=mx[x]; mx[x]=y; my[y]=x; y=yy;
          }
     }
     int maxMatching(){
          for (int x=0;x<n;x++){
               initBFS(x); int finish=-1;
               while (finish==-1){
                    finish=findPath(); if (finish!=-1) break;
                    finish=update(x);
               }
               enlarge(finish);
```

```
        }
        int total=0;
        for(int x=0;x<n;x++) if(cost[x][mx[x]]!=oo) total+=cost[x][mx[x]];
        return total;
    }
} match2;
void solve_match2(){
    int n,u,v,w; cin>>n; match2.init(n+1,n+1);
    while (cin>>u>>v>>w){ match2.add(u,v,w); }
    cout<<match2.maxMatching()<<"\n";
    for (int x=1,y;x<=n;x++){
        y=match2.mx[x]; if (y!=-1) cout<<x<<" "<<y<<"\n";
    }
}
```

Network flow (Edmonds Karp basic)

```
int n,m,source,target; vector<int> a[12309];
long c[2309][2309],f[2309][2309], d[12309];
const long oo = 1000111000111000;
void minimize(long &a, long b){ if (a>b) a=b; }
bool findpath(int start, int target){
    queue<int> qu; int i, u, v; for (i=1; i<=n; i++) d[i]=0;
    d[start] = oo; qu.push(start);
    while (qu.size()){
        u=qu.front(); qu.pop(); if (u==target) return true;
        for(i=0;v=a[u][i];i++) if(d[v]==0&&c[u][v]>f[u][v]){ d[v]=u; qu.push(v); }
    }
    return false;
}
void enlarge(){
    long u, v, delta=oo;
    for(v=target;(u=d[v])!=oo;v=u) minimize(delta,c[u][v]-f[u][v]);
    for(v=target;v!=source;v=u){ u=d[v]; f[u][v]+=delta; f[v][u]-=delta; }
}
long answer(int u){
    int i,v; long r=0; for(i=0;v=a[u][i];i++) r+=f[u][v];
    return r;
}
```

Network flow (Dinitz)

```
const int N = 1003, oo = 0x3c3c3c3c;
int n,m,S,T,d[N],c[N][N],f[N][N],Dfs[N],t=0; vector<int> a[N];
bool bfs(int S, int T) {
    memset(d, 0, sizeof d); queue<int> qu; qu.push(S); d[S]=1;
    while (qu.size()) {
        int u=qu.front(); qu.pop(); if (u==T) return true;
        for (int v: a[u])
            if (!d[v] && f[u][v]<c[u][v]) { qu.push(v); d[v]=d[u]+1; }
    }
    return false;
}
int visit(int u, int Min) {
    if (u==T) return Min;
    if (Dfs[u]!=t) Dfs[u]=t; else return 0;
    for (int v: a[u])
        if (f[u][v]<c[u][v]) if (Dfs[v]!=t && d[v]==d[u]+1)
        if (int x = visit(v, min(Min, c[u][v]-f[u][v])))
        { f[u][v]+=x; f[v][u]-=x; return x; }
    return 0;
}
```

Network flow basic

```
const int INF = 1000111000111000111LL;
struct Edge { int a, b, cap, flow, id;     };
struct MaxFlow {
    int n, s, t;    vector<int> d, ptr, q;
    vector< Edge > e;    vector< vector<int> > g;
    MaxFlow(int n) : n(n), d(n), ptr(n), q(n), g(n) {
        e.clear();  REP(i,n) {     g[i].clear(); ptr[i] = 0;    }
    }
    void addEdge(int a, int b, int cap, int id) {
        Edge e1 = { a, b, cap, 0, id };     Edge e2 = { b, a, 0, 0, id };
        g[a].push_back( (int) e.size() );   e.push_back(e1);
        g[b].push_back( (int) e.size() );   e.push_back(e2);
    }
    int getMaxFlow(int _s, int _t) {
        s = _s; t = _t;  int flow = 0;
        for (;;) {
            if (!bfs()) break;
            REP(i,n) ptr[i] = 0;
            while (int pushed = dfs(s, INF))flow += pushed;
        }
        return flow;
    }
    vector<int> trace() {
        bfs(); vector<int> res;
        for(auto edge : e) {
            if (d[edge.a] >= 0 && d[edge.b] < 0 && edge.cap > 0)
                res.push_back(edge.id);
        }
        return res;
    }
private:
    bool bfs() {
        int qh = 0, qt = 0;   q[qt++] = s;
        REP(i,n) d[i] = -1;
        d[s] = 0;
        while (qh < qt && d[t] == -1) {
            int v = q[qh++];
            REP(i,g[v].size()) {
                int id = g[v][i], to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;   d[to] = d[v] + 1;
                }
            }
        }
        return d[t] != -1;
    }
    int dfs (int v, int flow) {
```

```
        if (!flow) return 0;
        if (v == t) return flow;
        for (; ptr[v] < (int)g[v].size(); ++ptr[v]) {
            int id = g[v][ptr[v]],to = e[id].b;
            if (d[to] != d[v] + 1) continue;
            int pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
            if (pushed) {
                e[id].flow += pushed;   e[id^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
};
```

Network flow fastest

```
struct Edge {  int u, v, c, f;    int next; };
struct MaxFlow {
    int n, s, t;    vector< Edge > edges;
    vector<int> head, current, h, avail;
    vector<long long> excess;
    MaxFlow(int n) : n(n), head(n, -1), current(n, -1), h(n), avail(n), excess(n) {
        edges.clear();
    }
    int addEdge(int u, int v, int c) {
        Edge xuoi={u,v,c,0,head[u]};   head[u]=edges.size(); edges.push_back(xuoi);
        Edge nguoc={v,u,c,0,head[v]};  head[v]=edges.size(); edges.push_back(nguoc);
    }
    long long getMaxFlow(int _s, int _t) {
        s = _s; t = _t;  init();   int now = 0;   queue<int> qu[2];
        REP(x,n)    if (x != s && x != t && excess[x] > 0)qu[now].push(x);
        globalLabeling();    int cnt = 0;
        while (!qu[now].empty()) {
            while (!qu[1-now].empty()) qu[1-now].pop();
            while (!qu[now].empty()) {
                int x = qu[now].front(); qu[now].pop();
                while (current[x] >= 0) {
                    int p = current[x];
                    if (edges[p].c > edges[p].f && h[edges[p].u] > h[edges[p].v]) {
                        bool need=(edges[p].v!=s&&edges[p].v!=t&&!excess[edges[p].v]);
                        push(current[x]);    if (need) qu[1-now].push(edges[p].v);
                        if (!excess[x]) break;
                    }
                    current[x] = edges[current[x]].next;
                }
                if (excess[x] > 0) {
                    lift(x);   current[x] = head[x];   qu[1-now].push(x); cnt++;
                    if (cnt == n) {globalLabeling();  cnt=0;    }
                }
            }
            now = 1 - now;
        }
        return excess[t];
    }
private:
    void init() {
        REP(i,n) current[i] = head[i];
        int p = head[s];
        while (p >= 0) {
            edges[p].f = edges[p].c;    edges[p^1].f -= edges[p].c;
            excess[edges[p].v] += edges[p].c;    excess[s] -= edges[p].c;
            p = edges[p].next;
        }
        for(int v = 0; v < n; ++v) h[v] = 1;
        h[s] = n; h[t] = 0;
    }
    void push(int i) {
        long long delta = min(excess[edges[i].u], (long long) edges[i].c - edges[i].f);
        edges[i].f += delta; edges[i^1].f -= delta;
        excess[edges[i].u] -= delta;   excess[edges[i].v] += delta;
    }
    void lift(int u) {
        int minH = 2 * n;     int p = head[u];
        while (p>=0) {
            if (edges[p].c > edges[p].f)    minH = min(minH, h[edges[p].v]);
            p = edges[p].next;
        }
        h[u] = minH + 1;
    }
    void globalLabeling() {
        REP(i,n) avail[i] = 1, h[i] = 0;
        h[s] = n; h[t] = 0;   queue<int> q; q.push(t); avail[t] = false;
        while (!q.empty()) {
            int x = q.front(); q.pop();int p = head[x];
            while (p >= 0) {
                int pp = p^1;
                if (avail[edges[pp].u] && edges[pp].f < edges[pp].c) {
                    h[edges[pp].u] = h[x] + 1;    avail[edges[pp].u] = 0;
                    q.push(edges[pp].u);
                }
                p = edges[p].next;
            }
            if (q.empty() && avail[s]) {    avail[s] = false;  q.push(s);    }
        }
        REP(x,n) current[x] = head[x];
    }
};
```

Network flow Gomoryhu

```
struct GomoryHu {
    int ok[MN], cap[MN][MN], answer[MN][MN], parent[MN],n;    MaxFlow flow;
    GomoryHu(int n) : n(n), flow(n) {
        for(int i = 0; i < n; ++i) ok[i] = parent[i] = 0;
        for(int i = 0; i < n; ++i)for(int j = 0; j < n; ++j)
            cap[i][j] = 0, answer[i][j] = INF;
    }
    void addEdge(int u, int v, int c) {    cap[u][v] += c;    }
    void calc() {
```

```
        for(int i = 0; i < n; ++i) parent[i]=0;
        for(int i = 0; i < n; ++i)for(int j = 0; j < n; ++j)
            answer[i][j]=2000111000;
        for(int i = 1; i <= n-1; ++i) {
            flow = MaxFlow(n);
            REP(u,n) REP(v,n) if (cap[u][v]) flow.addEdge(u, v, cap[u][v]);
            int f = flow.getMaxFlow(i, parent[i]);    bfs(i);
            for(int j = i+1; j < n; ++j)
                if (ok[j] && parent[j]==parent[i])    parent[j]=i;
            answer[i][parent[i]] = answer[parent[i]][i] = f;
            for(int j = 0; j < i; ++j)
                answer[i][j]=answer[j][i]=min(f,answer[parent[i]][j]);
        }
    }
    void bfs(int start) {
        memset(ok,0,sizeof ok);   queue<int> qu;qu.push(start);
        while (!qu.empty()) {
            int u=qu.front(); qu.pop();
            for(int xid = 0; xid < flow.g[u].size(); ++xid) {
                int id = flow.g[u][xid];
                int v = flow.e[id].b, fl = flow.e[id].flow, cap = flow.e[id].cap;
                if (!ok[v] && fl < cap) {    ok[v]=1;  qu.push(v);    }
            }
        }
    }
};
```

Network flow mincut

```
pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();  VI used(N), cut, best_cut;   int best_weight = -1;
    for (int phase = N-1; phase >= 0; phase--) {
        VI w = weights[0];    VI added = used;   int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last; last = -1;
            for (int j = 1; j < N; j++)
            if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
            if (i == phase-1) {
                for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
                for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
                used[last] = true; cut.push_back(last);
                if (best_weight == -1 || w[last] < best_weight) {
                    best_cut = cut;best_weight = w[last];
                }
            }
            else {
                for (int j = 0; j < N; j++)  w[j] += weights[last][j];
                added[last] = true;
            }
        }
    }
    return make_pair(best_weight, best_cut);
}
```

Min cost basic

```
#define _MAX_COST INT_MAX
#define _MAX_FLOW INT_MAX
template<class Flow = int, class Cost = int>
struct MinCostFlow {
    struct Edge {
        int t; Flow f;   Cost c;   Edge*next, *rev;
        Edge(int _t, Flow _f, Cost _c, Edge*_next) :
                t(_t), f(_f), c(_c), next(_next) {}
    };
    vector<Edge*> E;
    int addV() {    E.push_back((Edge*) 0); return E.size() - 1;    }
    Edge* makeEdge(int s,int t,Flow f,Cost c){  return E[s]=new Edge(t,f,c,E[s]);}
    Edge* addEdge(int s, int t, Flow f, Cost c) {
        Edge*e1 = makeEdge(s, t, f, c), *e2 = makeEdge(t, s, 0, -c);
        e1->rev = e2, e2->rev = e1;    return e1;
    }
    pair<Flow, Cost> minCostFlow(int vs, int vt) {
        int n = E.size();    Flow flow = 0;Cost cost = 0;
        const Cost MAX_COST = _MAX_COST;    const Flow MAX_FLOW = _MAX_FLOW;
        for (;;) {
            vector<Cost> dist(n, MAX_COST); vector<Flow> am(n, 0);
            vector<Edge*> prev(n);vector<bool> inQ(n, false);  queue<int> que;
               dist[vs] = 0;  am[vs] = MAX_FLOW; que.push(vs);  inQ[vs] = true;
            while (!que.empty()) {
                int u=que.front(); Cost c=dist[u];    que.pop();    inQ[u]=false;
                for (Edge*e = E[u]; e; e = e->next)
                    if (e->f > 0) {
                        Cost nc = c + e->c;
                        if (nc < dist[e->t]) {
                            dist[e->t] = nc;  prev[e->t] = e;
                            am[e->t] = min(am[u], e->f);
                            if (!inQ[e->t]){  que.push(e->t);    inQ[e->t] = true;  }
                        }
                    }
            }
            if (dist[vt] == MAX_COST)  break;
            Flow by = am[vt]; int u = vt;   flow += by;   cost += by * dist[vt];
            while (u != vs) {
                Edge*e = prev[u];  e->f -= by;   e->rev->f += by;   u = e->rev->t;
            }
        }
        return make_pair(flow, cost);
    }
};
```

Min cost Dijkstra

```
#define F_INF 1000111000LL
#define C_INF 1000111000LL
template<class Flow = long long, class Cost = long long>
struct MinCostFlow {
    int V,E;   vector<Flow> cap;  vector<Cost> cost; vector<int> to, prev;
    vector<Cost> dist, pot;  vector<int> last, path, used;
    priority_queue< pair<Cost, int> > q;
    MinCostFlow(int  V,  int  E)  :  V(V),  E(0),  cap(E*2,0),  cost(E*2,0),  to(E*2,0),
prev(E*2,0),
```

```
        dist(V,0), pot(V,0), last(V, -1), path(V,0), used(V,0) {}
    int addEdge(int x, int y, Flow f, Cost c) {
        cap[E] = f; cost[E] = c; to[E] = y; prev[E] = last[x]; last[x] = E; E++;
        cap[E] = 0; cost[E] = -c; to[E] = x; prev[E] = last[y]; last[y] = E; E++;
        return E - 2;
    }
    pair<Flow, Cost> search(int s, int t) {
        Flow ansf = 0; Cost ansc = 0;  REP(i,V) used[i] = false;
        REP(i,V) dist[i] = C_INF;
        dist[s] = 0; path[s] = -1; q.push(make_pair(0, s));
        while (!q.empty()) {
            int x = q.top().second; q.pop();
            if (used[x]) continue; used[x] = true;
            for(int e = last[x]; e >= 0; e = prev[e]) if (cap[e] > 0) {
                Cost tmp = dist[x] + cost[e] + pot[x] - pot[to[e]];
                if (tmp < dist[to[e]] && !used[to[e]]) {
                    dist[to[e]] = tmp;  path[to[e]] = e;
                    q.push(make_pair(-dist[to[e]], to[e]));
                }
            }
        }
        REP(i,V) pot[i] += dist[i];
        if (used[t]) {
            ansf = F_INF;
            for(int e=path[t]; e >= 0; e=path[to[e^1]]) ansf = min(ansf, cap[e]);
            for(int e=path[t]; e >= 0; e=path[to[e^1]]) { ansc += cost[e] * ansf; cap[e]
-= ansf; cap[e^1] += ansf; }
        }
        return make_pair(ansf, ansc);
    }
    pair<Flow, Cost> minCostFlow(int s, int t) {
        Flow ansf = 0; Cost ansc = 0;
        while (1) {
            pair<Flow, Cost> p = search(s, t);   if (!used[t]) break;
            ansf += p.first; ansc += p.second;
        }
        return make_pair(ansf, ansc);
    }
};
```

Min cost SPFA

```
template<class Flow=int, class Cost=int>
struct MinCostFlow {
    const Flow INF_FLOW = 1000111000;  const Cost INF_COST = 1000111000111000LL;
    int n,t,S,T;Flow totalFlow;Cost totalCost;vector<int> last,visited;vector<Cost> dis;
    struct Edge {
        int to;Flow cap; Cost cost;     int next;
        Edge(int to,Flow cap,Cost cost,int next):to(to),cap(cap),cost(cost),next(next){}
    };
    vector<Edge> edges;
    MinCostFlow(int    n):n(n),t(0),totalFlow(0),totalCost(0),last(n,    -1),visited(n,
0),dis(n, 0){
        edges.clear();
    }
    int addEdge(int from, int to, Flow cap, Cost cost) {
        edges.push_back(Edge(to, cap, cost, last[from])); last[from] = t++;
        edges.push_back(Edge(from, 0, -cost, last[to]));  last[to] = t++;
        return t - 2;
    }
    pair<Flow, Cost> minCostFlow(int _S, int _T) {
        S = _S; T = _T;  SPFA();
        while (1) {
            while (1) {  REP(i,n) visited[i] = 0;     if (!findFlow(S, INF_FLOW)) break;
     }
            if (!modifyLabel()) break;
        }
        return make_pair(totalFlow, totalCost);
    }
private:
    void SPFA() {
        REP(i,n) dis[i] = INF_COST;    priority_queue< pair<Cost,int> > Q;
        Q.push(make_pair(dis[S]=0, S));
        while (!Q.empty()) {
            int x = Q.top().second;    Cost d = -Q.top().first;     Q.pop();
            if (dis[x] != d) continue;// For double: dis[x] > d + EPS
            for(int it = last[x]; it >= 0; it = edges[it].next)
                if (edges[it].cap > 0 && dis[edges[it].to] > d + edges[it].cost)
                    Q.push(make_pair(-(dis[edges[it].to]   =   d   +   edges[it].cost),
edges[it].to));
        }
        Cost disT = dis[T]; REP(i,n) dis[i] = disT - dis[i];
    }
    Flow findFlow(int x, Flow flow) {
        if (x == T) {    totalCost += dis[S] * flow;  totalFlow += flow; return   flow;
     }
        visited[x] = 1;  Flow now = flow;
        for(int it = last[x]; it >= 0; it = edges[it].next)
            if
(edges[it].cap&&!visited[edges[it].to]&&dis[edges[it].to]+edges[it].cost==dis[x]){
                Flow tmp = findFlow(edges[it].to, min(now, edges[it].cap));
                edges[it].cap -= tmp;   edges[it ^ 1].cap += tmp;    now -= tmp;
                if (!now) break;
            }
        return flow - now;
    }
    bool modifyLabel() {
        Cost d = INF_COST;
        REP(i,n) if (visited[i])
            for(int it = last[i]; it >= 0; it = edges[it].next)
                if (edges[it].cap && !visited[edges[it].to])
                    d = min(d, dis[edges[it].to] + edges[it].cost - dis[i]);
        if (d == INF_COST) return false;//For double:if(d>INF_COST/10)INF_COST = 1e20
        REP(i,n) if (visited[i])  dis[i] += d;
        return true;
    }
};
```

Bipartite graph

```
void check_bipartite(int u){
     visit[u]=1;
     for (int i=0,v;i<a[u].size();i++){
          v=a[u][i];
          if (!visit[v]){ color[v]=3-color[u]; check_bipartite(v); }
          else { if (color[u]==color[v]) invalid=1; }
     }
}
```

Cut vertex bridge

```
void visit(int u,int p){
     int num_child=0,v; num[u]=low[u]=++cnt;
     for (int i=0;i<a[u].size();i++){
          v=a[u][i];
          if (v!=p){
               if (num[v]) low[u]=min(low[u],num[v]);
               else {
                    visit(v,u); num_child++; low[u]=min(low[u],low[v]);
                    if (low[v]>=num[v]){ bridge.push_back(II(u,v)); }
                    if (u==p){ if (num_child>=2) cutnode[u]=1; }
                    else { if (low[v]>=num[u]) cutnode[u]=1; }
               }
          }
     }
}
```

Maximum clique

```
class MaxClique{
public:
    int n,el[maxn][log2maxn],s[maxn][log2maxn],dp[maxn],ans;
    vector<int> sol;
    void init(int v){
        n=v;ans=0; memset(el,0,sizeof(el)); memset(dp,0,sizeof(dp));
    }
    void addEdge(int u,int v){
        if (u>v) swap(u,v);
        if (u==v) return;
        el[u][v/32]|=(1<<(v%32));
    }
    bool dfs(int v,int k){
        int c=0,d=0,t=(n+31)/32;
        for (int i=0;i<t;i++){
            s[k][i]=el[v][i]; if (k!=1) s[k][i]&=s[k-1][i];
            c+=__builtin_popcount(s[k][i]);
        }
        if (c==0){
            if (k>ans){ ans=k; sol.clear(); sol.push_back(v); return 1; }
            return 0;
        }
        for (int i=0;i<t;i++){
            for (int a=s[k][i];a;d++){
                if (k+(c-d)<=ans) return 0;
                int lb=a&(-a),lg=0; a^=lb;
                while (lb!=1){ lb=(unsigned int)(lb)>>1; lg++; }
                int u=i*32+lg; if (k+dp[u]<=ans) return 0;
                if (dfs(u,k+1)){ sol.push_back(v); return 1; }
            }
        }
        return 0;
    }
    int solve(){
        for (int i=n-1;i>=0;i--){ dfs(i,1); dp[i]=ans; } return ans;
    }
}mc;
```

Steiner minimal trees

```
const int maxn = 30; const int oo = 1000111000;
struct SteinerTree {
     int n,a[maxn][maxn]; vector<int> s; vector<vector<int> > dp;
     void init(int _n){
          n=_n;
          for (int i=0;i<n;i++) for (int j=0;j<n;j++) a[i][j]=oo;
          for (int i=0;i<n;i++) a[i][i]=0;
          s.clear();
     }
     void addEdge(int u,int v,int w){
          a[u][v]=min(a[u][v],w); a[v][u]=min(a[v][u],w);
     }
     void addTerminal(int t){ s.push_back(t); }
     int minLengthSteinerTree(){
          int m=s.size(); if (m<=1) return 0;
          for (int k=0;k<n;k++) for (int i=0;i<n;i++) for (int j=0;j<n;j++)
               a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
          dp.resize(1<<m,vector<int>(n,0));
          for (int i=0;i<m;i++) for (int j=0;j<n;j++)
               dp[1<<i][j]=a[s[i]][j];
          for (int i=1;i<(1<<m);i++)
          if (((i-1)&i)!=0){
               for (int j=0;j<n;j++){
                    dp[i][j]=oo;
                    for (int k=(i-1)&i;k>0;k=(k-1)&i)
                         dp[i][j]=min(dp[i][j],dp[k][j]+dp[i^k][j]);
               }
               for (int j=0;j<n;j++) for (int k=0;k<n;k++)
                    dp[i][j]=min(dp[i][j],dp[i][k]+a[k][j]);
          }
          return dp[(1<<m)-1][s[0]];
     }
}steiner;
```

Strongly connected component (Tarjan)

```
void visit(int u){
     num[u]=low[u]=++cnt; st.push(u); int v;
     for (int i=0;i<a[u].size();i++){
          v=a[u][i]; if (num[v]) low[u]=min(low[u],num[v]);
          else { visit(v); low[u]=min(low[u],low[v]); }
     }
     if (num[u]==low[u]){
          scc++;
          do {
```

```
                v=st.top(); st.pop();
                num[v]=low[v]=oo; //TJAlG
//              num[v]=low[v]=-oo; //MESSAGE
          } while (v!=u);
     }
}
```

Topological sort

```
int n; vector<int> a[maxn];
bool vi[maxn]; vector<int> t;
void toposortDFS(int u){
     vi[u]=true;
     for(int i=0;i<a[u].size();i++) if(!vi[a[u][i]]){ toposortDFS(a[u][i]); }
     t.push_back(u);
}
void toposortDEG(int u){
     int c[maxn]={}; stack<int> st; while(st.size()) st.pop();
     for (int i=1;i<=n;i++)
     for (int j=0;j<a[i].size();j++){ c[a[i][j]]++; }
     for (int i=1;i<=n;i++) if (c[i]==0){ st.push(i); }
     while (st.size()){
          int u=st.top(); st.pop(); t.push_back(u);
          for (int i=0;i<a[u].size();i++){
               c[a[u][i]]--; if (c[a[u][i]]==0) st.push(a[u][i]);
          }
     }
     reverse(t.begin(),t.end());
}
```

Vertex cover

```
struct match_1{
     int nx,ny,m,adj[maxm],next[maxm],last[maxn],
     matx[maxn],maty[maxn],que[maxn],level[maxn],run[maxn];
     void init(int _nx,int _ny){
          nx=_nx; ny=_ny; m=0;
          for (int i=1;i<=nx;i++) matx[i]=-1,last[i]=-1;
          for (int i=1;i<=ny;i++) maty[i]=-1;
     }
     void add(int u,int v){ adj[m]=v; next[m]=last[u]; last[u]=m++; }
     bool bfs(){
          bool found=0; int size=0;
          for (int x=1;x<=nx;x++){
               if (matx[x]!=-1) level[x]=-1;
               else { level[x]=0; que[size++]=x; }
          }
          for(int i=0;i<size;i++){
               for(int x=que[i],e=last[x],y;e!=-1;e=next[e]){
                    y=adj[e];
                    if (maty[y]==-1) found=1;
                    else if (level[maty[y]]==-1){
                         level[maty[y]]=level[x]+1; que[size++]=maty[y];
                    }
               }
          }
          return found;
     }
     int dfs(int x){
          for (int &e=run[x],y;e!=-1;e=next[e]){
               y=adj[e];
               if(maty[y]==-1||(level[maty[y]]==level[x]+1&&dfs(maty[y]))){
                    maty[y]=x; matx[x]=y; return 1;
               }
          }
          return 0;
     }
     int maxMatching(){
          int total=0;
          while (bfs()){
               for (int x=1;x<=nx;x++) run[x]=last[x];
               for (int x=1;x<=nx;x++) if (matx[x]==-1) total+=dfs(x);
          }
          return total;
     }
} match1;
struct vertex_cover{
     int n,m; vector<int> a[maxn]; bool choosed[maxn];
     void init(int _n,int _m){
          n=_n; m=_m;
          for(int i=1;i<=n;i++){ a[i].clear(); choosed[i]=0; }
          match1.init(n,m);
     }
     void add(int u,int v){
          a[u].push_back(v); match1.add(u,v);
     }
     void konig(int u){
          choosed[u]=1;
          for (int i=0,v,x;i<a[u].size();i++){
               v=a[u][i];x=match1.maty[v];
               if (x!=-1 && x!=oo){
                    match1.maty[v]=oo; if (!choosed[x]) konig(x);
               }
          }
     }
     void solve(){
          cout<<match1.maxMatching();
          for (int i=1;i<=n;i++) if (match1.matx[i]==-1) konig(i);
          for (int i=1;i<=n;i++)
               if (match1.matx[i]!=-1 && !choosed[i]) cout<<" r"<<i;
          for (int j=1;j<=m;j++) if (match1.maty[j]==oo) cout<<" c"<<j;
          cout<<"\n";
     }
} cover;
```

DIE

```
const int rotations[6][4]={{1, 4, 0, 5},{1, 5, 0, 4},{4, 3, 5, 2},{4, 2, 5, 3},};
struct Die; map<int, Die> dieMap;
struct Die {
     int arr[6]; Die(){ for (int i=0;i<6;i++) arr[i]=i; }
     Die(int cipher) { // 0 -> 23
```

```
          if (dieMap.empty()) puts("Call openDie(die());");
          else (*this) = dieMap[cipher];
     }
     Die move(int dir) {
          Die res = (*this); int t = res.arr[rotations[dir][0]];
          for(int i=0;i<3;i++)
               res.arr[rotations[dir][i]]=res.arr[rotations[dir][i+1]];
          res.arr[rotations[dir][3]] = t;
          return res;
     }
     int encrypt() { // 0 -> 23
          int res=arr[0]*4; for(int i=3;i<=5;i++) if (arr[i]<arr[2]) res++;
          return res;
     }
     void print(){ for (int i=0;i<6;i++) cout<<arr[i]<<" "; cout<<"\n"; }
};
void openDie(Die t) {
     dieMap[t.encrypt()] = t;
     for (int dir=0;dir<4;dir++)
          if (!dieMap.count(t.move(dir).encrypt())) openDie(t.move(dir));
}
```

FFT

```
int my_round(double x) {
    if (x < 0) return -my_round(-x);
    return (int) (x + 1e-3);
}
const double PI = acos((double) -1.0); typedef complex<double> cplex;
int rev[MN]; cplex wlen_pw[MN], fa[MN], fb[MN];
void fft(cplex a[], int n, bool invert) {
    for (int i = 0; i < n; ++i) if (i < rev[i]) swap (a[i], a[rev[i]]);
    for (int len = 2; len <= n; len <<= 1) {
        double alpha = 2 * PI / len * (invert ? -1 : +1); int len2 = len >> 1;
        wlen_pw[0] = cplex(1, 0); cplex wlen(cos(alpha), sin(alpha));
        for (int i = 1; i < len2; ++i) wlen_pw[i] = wlen_pw[i-1] * wlen;
        for (int i = 0; i < n; i += len) {
            cplex t,*pu=a+i,*pv=a+i+len2,*pu_end=a+i+len2,*pw=wlen_pw;
            for (; pu != pu_end; ++pu, ++pv, ++pw) {
                t = *pv * *pw; *pv = *pu - t; *pu += t;
            }
        }
    }
    if (invert) REP(i, n) a[i] /= n;
}
void calcRev(int n, int logn) {
    REP(i,n){ rev[i]=0; REP(j, logn) if(i&(1<<j)) rev[i]|=1<<(logn-1-j); }
}
void mulpoly(int a[], int b[], ll c[], int na, int nb, int &n) {
    int l = max(na, nb), logn = 0; for (n = 1; n < l; n <<= 1) ++logn;
    n <<= 1; ++logn; calcRev(n, logn);
    REP(i,n) fa[i] = fb[i] = cplex(0);
    REP(i,na) fa[i] = cplex(a[i]);
    REP(i,nb) fb[i] = cplex(b[i]);
    fft(fa, n, false); fft(fb, n, false);
    REP(i,n) fa[i] *= fb[i];
    fft(fa, n, true); REP(i,n) c[i] = (ll)(fa[i].real() + 0.5);
}
```

Fraction

```
Fraction(double f){
     double integral = floor(f); double frac = f-integral;
     const T precision = 1000000000;
     T gcd_ = __gcd((T)round(frac*precision),precision);
     T denominator = precision/gcd_; T numerator = round(frac*precision)/gcd_;
     a=integral*denominator+numerator; b=denominator; norm();
}
```

Karatsuba

```
// - n must be power of 2. Remember to memset a, b, c to 0
ll buf[10000000],*ptr=buf;
void mul( int n, ll *a, ll *b, ll *c ) {
    if ( n<=32 ) {
        REP(i,2*n) c[i]=0;
        REP(i,n) REP(j,n) c[i+j]+=a[i]*b[j];
        REP(i,2*n) c[i]%=MOD;
        return;
    }
    int m=n/2; ll *s1=ptr; ptr+=n; ll *s2=ptr; ptr+=n;
    ll *s3=ptr; ptr+=n; ll *aa=ptr; ptr+=m; ll *bb=ptr; ptr+=m;
    REP(i,m) {
        aa[i]=a[i]+a[i+m]; bb[i]=b[i]+b[i+m];
        if ( aa[i]>=MOD ) aa[i]-=MOD;
        if ( bb[i]>=MOD ) bb[i]-=MOD;
    }
    mul(m,a,b,s1); mul(m,a+m,b+m,s2); mul(m,aa,bb,s3);
    memcpy(c,s1,n*sizeof(ll)); memcpy(c+n,s2,n*sizeof(ll));
    REP(i,n) c[i+m]+=s3[i]-s1[i]-s2[i];
    REP(i,2*n) c[i]%=MOD;
    ptr-=4*n;
}
// mul(2^x, a, b, c); REP(i,2*n) c[i] = (c[i] % MOD + MOD) % MOD
```

Lehmer pi

```
const int MAX = 2e6 + 5; const int M = 7;
vector<int> lp, primes, pi; int phi[MAX+1][M+1], sz[M+1];
void factor_sieve() {
    lp.resize(MAX); pi.resize(MAX); lp[1] = 1; pi[0] = pi[1] = 0;
    for (int i = 2; i < MAX; ++i) {
        if (lp[i] == 0) { lp[i] = i; primes.emplace_back(i); }
        for (int j = 0; j < primes.size() && primes[j] <= lp[i]; ++j) {
            int x = i * primes[j]; if (x >= MAX) break;
            lp[x] = primes[j];
        }
        pi[i] = primes.size();
    }
}
void init() {
    factor_sieve(); for(int i = 0; i <= MAX; ++i) { phi[i][0] = i; }
    sz[0] = 1;
```

```
    for(int i = 1; i <= M; ++i) {
        sz[i] = primes[i-1]*sz[i-1];
        for(int j=1;j<=MAX;++j){ phi[j][i]=phi[j][i-1]-phi[j/primes[i-1]][i-1]; }
    }
}
int sqrt2(long long x) {
    long long r = sqrt(x - 0.1); while (r*r <= x) ++r; return r - 1;
}
int cbrt3(long long x) {
    long long r = cbrt(x - 0.1); while(r*r*r <= x) ++r; return r - 1;
}
long long getphi(long long x, int s) {
    if(s == 0)  return x;
    if(s <= M){ return phi[x%sz[s]][s] + (x/sz[s])*phi[sz[s]][s]; }
    if(x <= primes[s-1]*primes[s-1]) { return pi[x] - s + 1; }
    if(x <= primes[s-1]*primes[s-1]*primes[s-1] && x < MAX){
        int sx=pi[sqrt2(x)]; long long ans=pi[x]-(sx+s-2)*(sx-s+1)/2;
        for(int i = s+1; i <= sx; ++i){ ans += pi[x/primes[i-1]]; }
        return ans;
    }
    return getphi(x, s-1) - getphi(x/primes[s-1], s-1);
}
long long getpi(long long x) {
    if(x < MAX)   return pi[x];
    int cx=cbrt3(x), sx=sqrt2(x); long long ans=getphi(x,pi[cx])+pi[cx]-1;
    for(int i=pi[cx]+1,ed=pi[sx];i<=ed;++i){ ans-=getpi(x/primes[i-1-1])-i+1; }
    return ans;
}
long long lehmer_pi(long long x) {
    if(x < MAX)   return pi[x];
     int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x)); int c = (int)lehmer_pi(cbrt3(x));
    long long sum = getphi(x, a) + (long long)(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++) {
        long long w=x/primes[i-1]; sum-=lehmer_pi(w); if(i > c) continue;
        long long lim = lehmer_pi(sqrt2(w));
        for(int j=i;j<=lim;j++){  sum-=lehmer_pi(w/primes[j-1])-(j - 1);  }
    }
    return sum;
}
```

Magic

```
void magic(int D[maxn],int K) {
     for(int b=0; b<K; b++) {
          int f=0, t=(1<<b);
          for(int g=0; g<(1<<(K-1-b));f+=(1<<(b+1)),t+=(1<<(b+1)),g++)
               for(int i=0; i<(1<<b); i++)  D[t|i] += D[f|i];//D[f|i] += D[t|i];
     }
}
```

BigNum

```
const int base = 1000000000,base_digits = 9;
struct BigInt {
    vector<int> a;  int sign;
    BigInt() : sign(1) {}
    BigInt(long long v) {    *this = v;    }
    BigInt(const string &s) {read(s);  }
    void operator=(const BigInt &v) {  sign = v.sign; a = v.a;  }
    void operator=(long long v) {
        sign = 1;   if (v < 0)    sign = -1, v = -v;
        a.clear();
        for (; v > 0; v = v / base)    a.push_back(v % base);
    }
    BigInt operator+(const BigInt &v) const {
        if (sign == v.sign) {
            BigInt res = v;
            for (int i=0,carry=0;i<(int)max(a.size(),v.a.size())||carry;++i){
                if (i == (int) res.a.size()) res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }
    BigInt operator-(const BigInt &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                BigInt res = *this;
                for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
                    res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
                    carry = res.a[i] < 0;    if (carry)    res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }
    void operator*=(long long v) {
        if (v < 0)  sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
            if (i == (int) a.size())    a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / base); a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
    }
    BigInt operator*(long long v) const {
        BigInt res = *this;   res *= v; return res;
    }
    friend pair<BigInt, BigInt> divmod(const BigInt &a1, const BigInt &b1) {
        int norm = base / (b1.a.back() + 1);
        BigInt a = a1.abs() * norm;    BigInt b = b1.abs() * norm;  BigInt q, r;
```

```
        q.a.resize(a.a.size());
        for (int i = a.a.size() - 1; i >= 0; i--) {
            r *= base;   r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2) / b.a.back(); r -= b * d;
            while (r < 0)     r += b, --d;
            q.a[i] = d;
        }
        q.sign = a1.sign * b1.sign;    r.sign = a1.sign;  q.trim(); r.trim();
        return make_pair(q, r / norm);
    }
    friend BigInt sqrt(const BigInt &a1) {
        BigInt a = a1;
        while (a.a.empty() || a.a.size() % 2 == 1)    a.a.push_back(0);
        int n = a.a.size();
        int firstDigit = (int) sqrt((double) a.a[n - 1] * base + a.a[n - 2]);
        int norm = base / (firstDigit + 1); a *= norm;     a *= norm;
        while (a.a.empty() || a.a.size() % 2 == 1)    a.a.push_back(0);
        BigInt r = (long long) a.a[n - 1] * base + a.a[n - 2];
        firstDigit = (int) sqrt((double) a.a[n - 1] * base + a.a[n - 2]);
        int q = firstDigit;  BigInt res;
        for(int j = n / 2 - 1; j >= 0; j--) {
            for(; ; --q) {
                BigInt r1 = (r - (res * 2 * base + q) * q) * base * base + (j > 0 ? (long
long) a.a[2 * j - 1] * base + a.a[2 * j - 2] : 0);
                if (r1 >= 0) {r = r1;   break;    }
            }
            res *= base; res += q;
            if (j > 0) {
                int d1 = res.a.size() + 2 < r.a.size() ? r.a[res.a.size() + 2] : 0;
                int d2 = res.a.size() + 1 < r.a.size() ? r.a[res.a.size() + 1] : 0;
                int d3 = res.a.size() < r.a.size() ? r.a[res.a.size()] : 0;
                q = ((long long) d1 * base * base + (long long) d2 * base + d3) / (firstDigit
* 2);
            }
        }
        res.trim(); return res / norm;
    }
    BigInt operator/(const BigInt &v) const {    return divmod(*this, v).first;    }
    BigInt operator%(const BigInt &v) const {    return divmod(*this, v).second;   }
    void operator/=(long long v) {
        assert(v < base);
        if (v < 0)  sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long) base;
            a[i] = (int) (cur / v);     rem = (int) (cur % v);
        }
        trim();
    }
    BigInt operator/(long long v) const {
        BigInt res = *this;   res /= v; return res;
    }
    int operator%(long long v) const {
        assert(v < base);
        if (v < 0)  v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)  m = (a[i] + m * (long long) base) % v;
        return m * sign;
    }
    void operator+=(const BigInt &v) { *this = *this + v; }
    void operator-=(const BigInt &v) { *this = *this - v; }
    void operator*=(const BigInt &v) { *this = *this * v; }
    void operator/=(const BigInt &v) { *this = *this / v; }
    bool operator<(const BigInt &v) const {
        if (sign != v.sign)   return sign < v.sign;
        if (a.size() != v.a.size())    return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])    return a[i] * sign < v.a[i] * sign;
        return false;
    }
    bool operator>(const BigInt &v) const {return v < *this;  }
    bool operator<=(const BigInt &v) const {    return !(v < *this);    }
    bool operator>=(const BigInt &v) const {    return !(*this < v);    }
    bool operator==(const BigInt &v) const {    return !(*this < v) && !(v < *this);}
    bool operator!=(const BigInt &v) const {    return *this < v || v < *this;    }
    void trim() {
        while (!a.empty() && !a.back())     a.pop_back();
        if (a.empty())   sign = 1;
    }
    bool isZero() const {    return a.empty() || (a.size() == 1 && !a[0]);   }
    BigInt operator-() const {    BigInt res = *this;res.sign = -sign;  return     res;
     }
    BigInt abs() const {BigInt res = *this;res.sign *= res.sign;   return res;    }
    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--)  res = res * base + a[i];
        return res * sign;
    }
    friend BigInt gcd(const BigInt &a, const BigInt &b) {
     return b.isZero() ? a : gcd(b, a % b);
    }
    friend BigInt lcm(const BigInt &a, const BigInt &b) { return a / gcd(a, b) * b;}
    void read(const string &s) {
        sign = 1;   a.clear();     int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-')     sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
```

```
    }
    friend istream& operator>>(istream &stream, BigInt &v) {
        string s;   stream >> s;   v.read(s);     return stream;
    }
    friend ostream& operator<<(ostream &stream, const BigInt &v) {
        if (v.sign == -1 && !v.isZero())    stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }
    static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits)
{
        vector<long long> p(max(old_digits, new_digits) + 1);  p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++) p[i] = p[i - 1] * 10;
        vector<int> res; long long cur = 0; int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
            cur += a[i] * p[cur_digits];    cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[new_digits]));
                cur /= p[new_digits];   cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back()) res.pop_back();
        return res;
    }
     typedef vector<long long> VLL;
    static VLL karatsubaMultiply(const VLL &a, const VLL &b) {
        int n = a.size();     VLL res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }
        int k = n >> 1;
        VLL a1(a.begin(), a.begin() + k);   VLL a2(a.begin() + k, a.end());
        VLL b1(b.begin(), b.begin() + k);   VLL b2(b.begin() + k, b.end());
        VLL a1b1 = karatsubaMultiply(a1, b1);
        VLL a2b2 = karatsubaMultiply(a2, b2);
        for (int i = 0; i < k; i++)    a2[i] += a1[i];
        for (int i = 0; i < k; i++)    b2[i] += b1[i];
        VLL r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)  r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)  r[i] -= a2b2[i];
        for (int i = 0; i < (int) r.size(); i++) res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)  res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)  res[i + n] += a2b2[i];
        return res;
    }
    BigInt operator*(const BigInt &v) const {
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        VLL a(a6.begin(), a6.end());   VLL b(b6.begin(), b6.end());
        while (a.size() < b.size())    a.push_back(0);
        while (b.size() < a.size())    b.push_back(0);
        while (a.size() & (a.size() - 1))   a.push_back(0), b.push_back(0);
        VLL c = karatsubaMultiply(a, b);    BigInt res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits); res.trim();
        return res;
    }
};
string toString(BigInt n){
    string s; stringstream ss;   ss<<n; ss>>s;  return s;
}
string toString(int n){
    string s; stringstream ss;   ss<<n; ss>>s;  return s;
}
BigInt power(BigInt a,BigInt b){
    if (b.isZero()) return 1;
    if (b==1) return a;
    BigInt t = power(a,b/2);     t=t*t;
    if (b%2==1) return t*a; else return t;
}
BigInt randBigInt(int len){
    string s=toString(rand()%9+1);
    for (int i=0;i<len;i++) s+=toString(rand()%10);
    return BigInt(s);
}
```

Reverse of mod

```
LL extgcd(LL a, LL b, LL &x, LL &y){
    LL g=a; x=1; y=0; if(b!=0){ g=extgcd(b,a%b,y,x); y-=(a/b)*x; }
    return g;
}
LL mod_inv(LL a,LL m){ LL x,y; extgcd(a,m,x,y); return (m+x%m)%m; }
```

GCD LCM

```
LL gcd(LL a,LL b){ return (b==0)?a:gcd(b,a%b);  }
LL lcm(LL a,LL b){ return (b*(a/gcd(a,b)));     }
```

Matrix

```
template <class type>
struct matrix {
    vector<vector<type> > f; int m,n;
    matrix(int m=0,int n=0) : m(m), n(n) {
        f.resize(m);
        for (int i=0;i<m;i++){
            f[i].resize(n); for (int j=0;j<n;j++){ f[i][j]=0; }
        }
    }
```

```
     void operator = (const matrix &a);
     void print(){
          for(int i=0;i<m;i++){ for(int j=0;j<n;j++){ cout<<f[i][j]<<" "; } cout<<"\n"; }
          cout<<"\n";
     }
};
template<class type>
matrix<type> identity(int n){
     matrix<type> i(n,n); for (int j=0;j<n;j++) i.f[j][j]=1;
     return i;
}
template<class type>
matrix<type> operator * (const matrix<type> &a, const matrix<type> &b){
     matrix<type> c(a.m,b.n);
     for (int i=0;i<c.m;i++) for (int j=0;j<c.n;j++) for (int k=0;k<a.n;k++)
          c.f[i][j]=(c.f[i][j]+a.f[i][k]*b.f[k][j])%mod;
     return c;
}
template<class type>
matrix<type> operator ^ (const matrix<type> &a, int k){
     if (k==0) return identity<type>(a.n);
     if (k==1) return a;
     matrix<type> t=a^(k/2); t=t*t; if (k&1) t=t*a;
     return t;
}
template<class type>
void matrix<type> :: operator = (const matrix<type> &a){
     m=a.m; n=a.n; for (int i=0;i<m;i++) for (int j=0;j<n;j++) f[i][j]=a.f[i][j];
}
//Matrix_math
void getCofactor(int mat[N][N], int temp[N][N], int p, int q, int n){
     int i = 0, j = 0;
     for (int row = 0; row < n; row++) for (int col = 0; col < n; col++){
          if(row!=p&&col!=q){ temp[i][j++]=mat[row][col]; if(j==n-1){ j=0; i++; } }
     }
}
int determinantOfMatrix(int mat[N][N], int n){
     int D = 0; if (n == 1) return mat[0][0];
     int temp[N][N]; int sign = 1;
     for(int f = 0; f < n; f++){
          getCofactor(mat,temp,0,f,n);
          D+=sign*mat[0][f]*determinantOfMatrix(temp,n-1); sign=-sign;
     }
     return D;
}
void adjoint(int A[N][N],int adj[N][N]){
     if (N == 1){ adj[0][0] = 1; return; }
     int sign = 1, temp[N][N];
     for (int i=0; i<N; i++) for (int j=0; j<N; j++){
          getCofactor(A, temp, i, j, N); sign = ((i+j)%2==0)? 1: -1;
          adj[j][i] = (sign)*(determinant(temp, N-1));
     }
}
bool inverse(int A[N][N], double inverse[N][N]){
     int det = determinant(A, N); if (det == 0) return false;
     int adj[N][N]; adjoint(A, adj);
     for(int i=0;i<N;i++) for(int j=0;j<N;j++) inverse[i][j]=adj[i][j]/double(det);
     return true;
}
```

Cramer

```
LL det(int a1,int b1,int c1,int a2,int b2,int c2,int a3,int b3,int c3){
     return  +a1*(b2*c3-b3*c2)-b1*(a2*c3-a3*c2)+c1*(a2*b3-a3*b2);
}
void cramer(int a1,int b1,int c1,int d1,int a2,int b2,int c2,int d2,
               int a3,int b3,int c3,int d3){
     LL D = det(a1,b1,c1, a2,b2,c2, a3,b3,c3);
     double DX = det(d1,b1,c1, d2,b2,c2, d3,b3,c3);
     double DY = det(a1,d1,c1, a2,d2,c2, a3,d3,c3);
     double DZ = det(a1,b1,d1, a2,b2,d2, a3,b3,d3);
     if (D==0) cout<<"Math error\n"; else cout<<DX/D<<" "<<DY/D<<" "<<DZ/D<<"\n";
}
```

Diophantine

```
// a * x + b * y = c , c % gcd(a, b) == 0
int diophantine(int a,int b,int c,int &x,int &y){
     int g=extgcd(a,b,x,y); x=c/g*x; y=c/g*y; return g;
}
```

Gaussian

```
int gauss(vector<vector<double> > a,vector<double> &ans){
     int n=a.size(); int m=a[0].size()-1; vector<int> where(m,-1);
     for (int col=0,row=0;col<m && row<n;++col){
          int sel=row;
          for(int i=row;i<n;i++) if(abs(a[i][col])>abs(a[sel][col])) sel=i;
          if (abs(a[sel][col])<EPS) continue;
          for (int i=col;i<=m;i++) swap(a[sel][i],a[row][i]);
          where[col]=row;
          for (int i=0;i<n;i++)
          if (i!=row){
               double c=a[i][col]/a[row][col];
               for (int j=col;j<=m;j++) a[i][j]-=a[row][j]*c;
          }
          row++;
     }
     ans.assign(m,0);
     for(int i=0;i<m;i++) if(where[i]!=-1) ans[i]=a[where[i]][m]/a[where[i]][i];
     for (int i=0;i<n;i++){
          double sum=0; for (int j=0;j<m;j++) sum+=ans[j]*a[i][j];
          if (abs(sum-a[i][m])>EPS) return 0;
     }
     for (int i=0;i<m;i++) if (where[i]==-1) return oo;
     return 1;
}
int gauss(vector<bitset<maxn> > a,bitset<maxn> &ans){
     int n=a.size(); int m=a[0].size()-1; vector<int> where(m,-1);
     for (int col=0,row=0;col<m && row<n;++col){
          for (int i=row;i<n;i++)
```

```
          if (a[i][col]){ swap(a[i],a[row]); break; }
          if (!a[row][col]) continue;
          where[col]=row;
          for (int i=0;i<n;i++) if (i!=row && a[i][col]) a[i]^=a[row];
          row++;
     }
}
```

Binomial coefficients

```
const int maxn = 60; ULL BC[maxn][maxn];
void binomialCoeff(){
     BC[0][0]=1; for (int i=1;i<maxn;i++) for (int j=0;j<=i;j++)
          if (j==0) BC[i][j]=1; else BC[i][j]=BC[i-1][j-1]+BC[i-1][j];
}
ULL binomialCoeff(int n,int k){
     ULL res=1; if (k>=n-k) k=n-k;
     for (int i=0;i<k;i++){ res*=(n-i); res/=(i+1); }
     return res;
}
```

Chinese remainder

```
struct CRT {
     vector<LL> a,b,m; LL x,M;
     void init(){ a.clear(); b.clear(); m.clear(); }
     void add(LL b_, LL m_){
          a.push_back(1LL); b.push_back(b_); m.push_back(m_);
     }
     bool linearCongruences(){
          LL n=a.size(); x=0;; M=1;
          for (int i=0;i<n;i++){
               LL a_=a[i]*M, b_=b[i]-a[i]*x,m_=m[i];
               LL y,t,g=extgcd(a_,m_,y,t);
               if (b_%g) return false;
               b_/=g; m_/=g; x+=M*(y*b_%m_); M*=m_;
          }
          x=(x+M)%M; return true;
     }
}crt;
```

Prime factors

```
VLL primeFactors(LL x){
     VLL f;    LL i=0,n=primes[i];
     while(n*n<=x){while(x%n==0){x/=n;     f.push_back(n);     }    n=primes[++i];}
     if (x!=1) f.push_back(x);
     return f;
}
LL sumDiv(LL x){
     LL i=0,n=primes[i],ans=1;
     while (n*n<=x){
          int c=0;  while (x%n==0) { c++; x/=n; }
          ans*=(pow(n,c+1)-1)/(n-1);   n=primes[++i];
     }
     if (x!=1) ans*=(pow(x,2)-1)/(x-1);
     return ans;
}
```

```
LL EulerPhi(LL x){
     if (n==0) return 0;
     LL i=0,n=primes[i],phi=x;
     while (n*n<=x){
          if (x%n==0) phi-=phi/n;
          while (x%n==0) x/=n;
          n=primes[++i];
     }
     if (x!=1) phi-=phi/x;
     return phi;
}
LL f[maxn],p[maxn];
void init_EulerPhi(LL n){
     bool stop=false;
     if (!stop){
          for (int i=1;i<n;i++)   p[i]=1, f[i]=i;
          for (int i=2;i<n;i++){
               if (p[i]){
                    f[i]-=f[i]/i;  for(int j=i+i;j<n;j+=i) p[j]=0,f[j]-=f[j]/i;
               }
          }
          stop=true;
     }
}
```

Prime Phi

```
LL p[maxn][maxp];  LL phi[maxn]; LL phi_[maxn];
void phi_function(){
     for (int i=1;i<maxn;i++)     phi_[i]=i;
     for (int i=2;i<maxn;i++)
     if(phi_[i]==i)for (int j=i;j<maxn;j+=i)    phi_[j]=(phi_[j]/i)*(i-1);
}
LL power(int a,int k){
     if (k==0) return 1;
     if (p[a][k]>0)return p[a][k];
     LL t=power(a,k/2); t*=t;     if (k%2==1) t*=a;
     return p[a][k]=t;
}
LL phi_function(int p,int k){
     if (k==0) return 1;
     return (p-1)*power(p,k-1);
}
LL phi_function(int m){
     if (phi[m]>0) return phi[m];
     LL r=1,t=m,k;
     for (int i=2;i*i<=t;i++){
          k=0; while (t%i==0) k++,t/=i;
          r*=phi_function(i,k);
     }
     if (t>1) r*=phi_function(t,1);
     return phi[m]=r;
}
```

Prime Pollard Rho

```
LL f(LL x,LL m,LL k){   return (mul_mod(x,x,m)+k%m)%m;    }
```

```
LL pollard_rho(LL n,LL k){
     LL a=2, b=2;
     do {
          a=f(a,n,k);    b=f(b,n,k);    b=f(b,n,k);
          LL p=gcd(abs(b-a),n);   if (p>1) return p;
     } while (a!=b);
     return n;
}
LL get_factor(LL n){
     if (n==1) return n;
     if (isPrime(n)) return n;
     if (n==4) return 2;
     LL p=n,q;
     for (LL k=1;p==n;k++){  p=pollard_rho(n,k);q=n/p;    }
     return p;
}
```

Prime Rabin Miller

```
LL mulMod(LL x, LL y, LL p) {
     if (y == 0) return 0;
     if (x < 10001110000111000111LL / y) return x * y % p;
     LL mid = mulMod((x+x)%p, y>>1LL, p);
     if (y & 1) return (mid + x) % p;  else return mid;
}
LL powMod(LL x, LL k, LL m) {
     if (k == 0) return 1;
     if ((k & 1)) return mulMod(x,powMod(x, k-1, m), m);
     else return powMod(mulMod(x,x,m), k/2, m);
}
bool suspect(LL a, LL s, LL d, LL n) {
     LL x = powMod(a, d, n); if (x == 1) return true;
     for (int r = 0; r < s; ++r) {
          if (x == n - 1) return true;
          x = mulMod(x, x, n);
     }
     return false;
}
bool isPrime(LL n) {
     if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
     LL test[] = {2,3,5,7,11,13,17,19,23,-1};   LL d = n - 1, s = 0;
     while (d % 2 == 0) ++s, d /= 2;
     for (int i = 0; test[i] < n && test[i] != -1; ++i)
          if (!suspect(test[i], s, d, n)) return false;
     return true;
}
```

Prime Segmented sieve

```
const int SQRTN = 1<<16; // upperbound of sqrt(H) + 10
vector<bool> segmentSieve(LL L, LL H) {
     static LL p[SQRTN];static int lookup = 0;
     if (!lookup) {
          for (LL i = 2; i < SQRTN; ++i) p[i] = i;
          for (LL i = 2; i*i < SQRTN; ++i)
               if (p[i]) for (LL j = i*i; j < SQRTN; j += i)   p[j] = 0;
          remove(p, p+SQRTN, 0);  lookup = 1;
     }
     vector<bool> table(H - L);
     for (LL i = L; i < H; ++i) table[i - L] = 1;
     for (LL i = 0, j; p[i] * p[i] < H; ++i) { // O( \sqrt(H) )
          if (p[i] >= L) j = p[i] * p[i];
          else if (L % p[i] == 0) j = L;
               else j = L - (L % p[i]) + p[i];
          for (; j < H; j += p[i]) table[j-L] = 0;
     }
     return table;
}
```

Infix to postfix

```
int GetPriority(char c){
     if (c=='*' || c=='/') return 2;
     if (c=='+' || c=='-') return 1;
     if (c=='(') return 0;
}
string infix_postfix(string s){
     string res=""; stack<char> st;
     for (int i=0;i<s.size();i++){
          if (s[i]==' '){ continue; }
          if (s[i]=='('){ st.push(s[i]); continue; }
          if (s[i]==')'){
               while (!st.empty() && st.top()!='('){ res=res+st.top()+" "; st.pop(); }
               st.pop(); continue;
          }
          if (s[i]=='+' || s[i]=='-' || s[i]=='*' || s[i]=='/'){
               while (!st.empty() && GetPriority(st.top())>=GetPriority(s[i])){
                    res=res+st.top()+" "; st.pop();
               }
               st.push(s[i]); continue;
          }
          if ('0'<=s[i] && s[i]<='9'){
               while ('0'<=s[i] && s[i]<='9'){ res+=s[i]; i++; }
               i--; res+=" "; continue;
          }
     }
     while (!st.empty()){ res=res+st.top()+" "; st.pop(); }
     return res;
}
int calc_postfix(string s){
     stack<int> st;
     for (int i=0;i<s.size();i++){
          int x=0; if (s[i]==' ') continue;
          if ('0'<=s[i] && s[i]<='9'){
               while ('0'<=s[i] && s[i]<='9'){ x=x*10+(s[i]-'0'); i++; }
               i--;
          } else if (s[i]=='+'){
               x=st.top(); st.pop(); x+=st.top(); st.pop();
          }  else if (s[i]=='-'){
               x=st.top(); st.pop(); x-=st.top(); st.pop();
          }  else if (s[i]=='*'){
               x=st.top(); st.pop(); x*=st.top(); st.pop();
```

```
          } else if (s[i]=='/'){
               x=st.top(); st.pop(); x/=st.top(); st.pop();
          }
          st.push(x);
     }
     return st.top();
}
```

Josephus

```
LL josephus(LL n,LL k){
     LL p=n*k; LL nn=n+1, kk=k-1; while (p>n){ p+=(p-nn)/kk-n; }
     return p;
}
```

LIS

```
void solve(){
     for (int i=1;i<=n;i++){
          f[i]=lower_bound(b+1,b+ans+1,a[i])-b; ans=max(ans,f[i]); b[f[i]]=a[i];
     }
     int exp=ans;
     for (int i=n;i>=1;i--) if (f[i]==exp){ t.push_back(a[i]); exp--; }
     reverse(t.begin(),t.end());
}
```

Longest biosed interval

```
int n,l,x,y; long long a[maxn];
void LBI(){
     long long m[maxn]={},c[maxn]={}; x=0; y=0;
     for (int i=1;i<=n;i++){
          c[i]=c[i-1]+a[i];
          if (c[i-1]<c[m[i-1]]) m[i]=i-1; else m[i]=m[i-1];
          int k=i-y+x-1;
          while (k>0){
               if (c[i]-c[m[k]]>=l) k=m[k]; else break;
               x=k+1; y=i;
          }
     }
}
```

Max sat

```
int n,m,cnt;   vector<int> *a;     int *color,*num,*low;
stack<int> st; bool invalid=0;
#define SetLength(a,n,t) a=((t*)calloc(n,sizeof(t)))+(n)/2
void init(int m,int n){
     SetLength(a,n*2+10,vector<int>);  SetLength(color,n*2+10,int);
     SetLength(num,n*2+10,int);   SetLength(low,n*2+10,int);
}
void setColor(int u,int x){
     if (color[u]==(x^3)) invalid=1;   else color[u]=x;
     if (color[-u]==x) invalid=1; else color[-u]=(x^3);
}
void tarzan(int u){
     num[u]=low[u]=++cnt;     st.push(u);
     for (int i=0,v;i<a[u].size();i++){
          v=a[u][i];
          if (num[v]) low[u]=min(low[u],num[v]);
          else tarzan(v), low[u]=min(low[u],low[v]);
          if (color[v]==1) setColor(u,1); // set false
     }
     if (low[u]==num[u]){
          int v=0;  if (color[u]==0) setColor(u,2); // set true
          do {
               v=st.top(); st.pop();   setColor(v,color[u]);   num[v]=low[v]=oo;
          } while (u!=v);
     }
}
void input(){
     cin>>m>>n;     init(m,n);     int p,q;
     for (int i=1;i<=m;i++){
          cin>>p>>q;     a[-p].push_back(q);a[-q].push_back(p);
     }
}
void solve(){
     for (int i=1;i<=n;i++)  if (!num[i]) tarzan(i);
     for (int i=1;i<=n;i++)  if (!num[-i]) tarzan(-i);
}
void output(){
     if (invalid) cout<<"NO\n";
     else {
          cout<<"YES\n"; int ans=0;
          for (int i=1;i<=n;i++)  if (color[i]==2) ans++;
          cout<<ans<<"\n";
          for (int i=1;i<=n;i++)  if (color[i]==2)   cout<<i<<" ";
          cout<<"\n";
     }
}
```

Bitmask

```
void misc(){
     cout<<(n&(n-1))<<"\n";// set last one bit to 0
     cout<<(n&-n)<<"\n";// set all to 0 except for last one bit
     cout<<(n|(n-1))<<"\n";// invert all bit after last one bit
     cout<<(n&(n-1))<<"\n";// if power of two equal 0
     cout<<__builtin_clz(n)<<"\n";// the number of leading 0-bits in x
     cout<<__builtin_ctz(n)<<"\n";// the number of trailing 0-bits in x
     cout<<__builtin_popcount(n)<<"\n";// the number of 1-bits in x
     cout<<__builtin_parity(n)<<"\n";// the number of 1-bits in x modulo 2.
}
```

DP Digit

```
const int maxd = 25,maxt = 2;     LL dp[maxd][maxt];
unordered_map<LL,LL> mm;     vector<int> d;
struct DP_digit {
     void getDigit(LL n){
          d.clear();     while (n){     d.push_back(n%10); n/=10;    }
     }
     LL calc(int pos,bool tight,LL sum){
          if (pos==-1)  return sum;
          if (dp[pos][tight]!=-1) return dp[pos][tight];
          LL res=0; int k=(tight)?d[pos]:9; bool ntight;
          for (int i=0;i<=k;i++){
```

```
            ntight=(d[pos]==i)?tight:0;  res+=calc(pos-1,ntight,sum+i);
        }
        return res;
    }
    LL calc(LL u){
        if (mm.count(u)) return mm[u];
        memset(dp,-1,sizeof(dp));    getDigit(u);
        return mm[u]=calc(d.size()-1,1,0);
    }
    LL calc(LL u,LL v){return calc(v)-calc(u-1);    }
};
```

Aho corasick

```
namespace trie {
    const int N=2003; int a[N][128] Peak=0,Prev[N]; bool Leaf[N];
    void clear() {
        Peak=0; memset(a, 0, sizeof a);
        memset(Prev, 0, sizeof Prev); memset(Leaf, 0, sizeof Leaf);
    }
    void insert(char s[]) {
        int u=0;
        for (int i=0; char k=s[i]; i++) {
            if (!a[u][k]) a[u][k] = ++Peak;
            u=a[u][k];
        }
        Leaf[u]=true;
    }
    int next(int u, char k) {
        for (int i=u; i!=-1; i=Prev[i])
        if (a[i][k]) return a[i][k];
        return 0;
    }
    void bfs() {
        queue<int> qu; qu.push(0); Prev[0]=-1;
        while (qu.size()) {
            int u=qu.front(); qu.pop();
            for (int k=0; k<128; k++)
            if (int v=a[u][k]) {
                Prev[v] = next(Prev[u], k); Leaf[v] |= Leaf[Prev[v]]; qu.push(v);
            }
        }
    }
};
```

KMP

```
int pi[maxl],cnt[maxl];
void initKMP(string p){
    int m=p.size();
    for (int i=1,j=0;p[i];i++){
        while (j && (p[i]!=p[j]))    j=pi[j-1];
        if (p[i]==p[j]) pi[i]=++j;
    }
}
void getKMP(string s,string p){
    int n=s.size(); int m=p.size();
    for (int i=0,j=0;s[i];i++){
        while (j && (s[i]!=p[j]))    j=pi[j-1];
        if (s[i]==p[j]){
            cnt[++j]++;
            if (j==m){
                j=pi[j-1];//cout<<i-m+2<<" ";
            }
        }
    }
}
```

Lyndon

```
void lyndon(string s) {
    int n = (int) s.length(); int i = 0;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if(s[k]<s[j]) k=i; else ++k;
            ++j;
        }
        while (i <= k){    cout<<s.substr(i,j-k)<<' '; i+=j-k; }
    }
    cout << endl;
}
```

Manacher

```
const char DUMMY = '.';
int manacher(string s) {
    int n=s.size()*2-1; vector<int> f=vector<int>(n,0);
    string a=string(n,DUMMY); for(int i=0;i<n;i+=2) a[i]=s[i/2];
    int l = 0, r = -1, center, res = 0;
    for (int i = 0, j = 0; i < n; i++) {
        j=(i>r?0:min(f[l+r-i],r-i))+1; while(i-j>=0&&i+j<n&&a[i-j]==a[i+j]) j++;
        f[i] = --j; if (i + j > r){ r = i + j; l = i - j; }
        int len = (f[i] + i % 2) / 2 * 2 + 1 - i % 2;
        if (len > res){ res = len; center = i; }
    }
    return res;// a[center - f[center]..center + f[center]] is the needed substring
}
```

Minmove

```
int minmove(string s) {
    int n=s.length(); int x,y,i,j,u,v; // x is the smallest string before string y
    for (x = 0, y = 1; y < n; ++ y) {
        i = u = x; j = v = y;
        while (s[i] == s[j]) {
            ++ u; ++ v;
            if (++ i == n) i = 0;
            if (++ j == n) j = 0;
            if (i == x) break; // All strings are equal
        }
        if (s[i] <= s[j]) y = v; else{ x = y; if (u > y) y = u; }
    }
    return x;
}
```

Suffix array

```
struct SuffixArray {
    string a;  int N, m;
    vector<int> SA, LCP, x, y, w, c;
    SuffixArray(string _a, int m = 256) : a(" " + _a), N(a.length()), m(m), SA(N), LCP(N),
x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;   DA();     kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back(); }
        REF(SA); REF(LCP);    a = a.substr(1, a.size());
        for(int i = 0; i < (int) SA.size(); ++i) --SA[i];
        #undef REF
    }
    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y[b] && y[a
+ l] == y[b + l]); }
    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }
    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }
    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j + k]; k++);
            else k = 0;
    }
};
```

Z function

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

Trie

```
const int maxc = 256;
struct trie {
     struct node {
          int a[maxc];  int val;
          int& operator[] (int i){     return a[i%maxc];  }
          node(){   memset(a,0,sizeof(a)); val=0;     }
     };
     vector<node> a;
     int& operator[] (string s){
          int pos=0,c;
          for (int i=0;c=s[i];i++){
               if (a[pos][c]==0){ a.push_back(node()); a[pos][c]=a.size()-1; }
               pos=a[pos][c];
          }
          return a[pos].val;
     }
     void clear(){  a.clear(); a.push_back(node());  }
     trie(){   clear();  }
}tr;
```

Content

RABIN KARP

```
vector<int> rabin_karp(string const& s, string const& t)
{
    const int p = 31;
    const int mod = 1e9+9;

    vector<long long> p_pow(max(s.size(),t.size()));
    p_pow[0]=1;
    for (int i=1;i<p_pow.size();i++)
        p_pow[i]=(p_pow[i-1]*p)%mod;
    vector<long long> hash_T(t.size()+1,0);
    for (int i=0;i<t.size();i++)
    {
        hash_T[i+1]=(hash_T[i]+(t[i]-'a'+1)*p_pow[i])%mod;
    }
    long long hash_s=0;
    for (int i=0;i<s.size();i++)
    {
        hash_s=(hash_s+(s[i]-'a'+1)*p_pow[i])%mod;
    }
    vector<int> ocurrences;
    for (int i=0;(i+(int)s.size()-1)<t.size();i++)
    {
        long long cur_t =(hash_T[i+s.size()]+mod-hash_T[i])%mod;
        if ((hash_s*p_pow[i])%mod==cur_t)
            ocurrences.push_back(i);
    }
    return ocurrences;
}
```

HASH FUNCTION

```
// optionn 1 (slower - don't know why)
struct chash{
 ll operator()(ll x) const {return std::hash<ll>{}((x^RANDOM)%MOD *
MUL);}
};

// Option 2
struct chash{
   int operator()(int x) const{return x^RANDOM;}
};

// Option 3 (fastest)
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<int,int, chash> inQ;
```