Graph Theory

Bridge:

```
vector<int>vc[100001];
int vis[100001];
int cnt,in[100001],val[100001];
set< pair < int, int > > bridge;
void dfs(int v,int p) {
  vis[v] = 1;
  in[v] = val[v] = cnt++;
  REP(i, vc[v].size()) {
    if(vc[v][i] == p)continue;
    if(vis[vc[v][i]])val[v] = min(val[v], in[vc[v][i]]);
    else{
      dfs(vc[v][x], v);
      val[v] = min(val[v], val[vc[v][i]]);
      if (val[vc[v][x]] > in[v]){
        pair<int,int> pp = make_pair(v,vc[v][x]);
        if(pp.xx > pp.yy)swap(pp.xx, pp.yy);
                                  bridge.insert(pp);
        }
     }
  }
}
```

Articulation Point:

```
vector<int>vc[100001];
int vis[100001];
int cnt = 0,in[100001],val[100001];
set< int > apNodes;
void dfs(int v,int p){
vis[v] = 1;
in[v] = val[v] = cnt++;
int c=0;
REP(i, vc[v].size()){
  if(vc[v][i] == p)continue;
  if(vis[vc[v][i]])val[v] = min(val[v], in[vc[v][i]]);
  else{
    dfs(vc[v][x],v);
    val[v]=min(val[v], val[vc[v][i]]);
    if (val[vc[v][i]] >= in[v] && p!=-1)
        apNodes.insert(v);
    c++;
   }
}
if(p==-1 && c>1) apNodes.insert(v);
}
```

Euler's theorem. For any planar graph, V − E + F = 1 + C, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: V − E + F = 2 for a 3D polyhedron.

Vertex covers and independent sets. Let M , C, I be a max matching, a min vertex cover, and a max independent set. Then |M | |C| = N − |I|, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is I = (A ∩ S) ∪ (B ∩ T), and a minimum(-weighted) vertex cover is C = (A ∩ T) ∪ (B ∩ S).

Matrix-tree theorem. Let matrix T = [t ij], where t ij is the number of multiedges between i and j, for i = j, and t ii = −deg i . Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T .

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected.

Construction:

doit(u):

for each edge e = (u, v) in E, do: erase e, doit(v) prepend u to the list of vertices in the tour.

Havel Hakimi:

```
f = 0;
REP(j, b){
  sort(tmp + j, tmp + b);
  reverse(tmp + j, tmp + b);
  if(tmp[j] < 0){
     f = 1;
     break;
  }
  for(int k = 1; k <= tmp[j]; k++){
      if(j + k >= b){
          f = 1;
          break;
       }
       tmp[j + k]--; } }
// havel-hakimi ends here. f = 1 shows invalid graph
```

Stable Marriage:

```
scanf("%d", &n);
for( int i = 0; i < n; i++ ) {
        for( int j = 0; j < n; j++ ) {
                scanf("%d", &mPref[i][j]);
        }
}
for( int i = 0; i < n; i++ ) {
        for( int j = 0; j < n; j++ ) {
                int k;
                scanf("%d", &k);
                wPref[i][k-1] = n - j;
        }
}
memset( R, -1, sizeof(R) );
memset( P, 0, sizeof(P) );
for( int i = 0; i < n; i++ ) {
        int man = i;
        while( man >= 0 ) {
             int woman = mPref[man][ P[man]++ ];
      if( R[woman] == -1 || wPref[woman]
[ R[woman] ] < wPref[woman][ man ] ) {
        L[man] = woman;
        swap( man, R[woman] );
        }
      }
   }
printf("Case %d:", ++caseno);
for(int i=0;i<n; i++) printf("%d %d", i+1, L[i]+n+1);
```

FLOW

Dinic:

```
struct edge {
int a, b, cap, flow ;
edge(int _a,int _b,int _c,int _d) {
    a=_a,b=_b,cap=_c,flow=_d; } } ;
int INF = 1500000001 ;
int n,s,t,d[30001] , ptr [ 30001 ] , q [ 30001 * 10 ] ;
vector < edge > e,g [ 30001 ] ;
void add_edge ( int a, int b, int cap ) {
edge e1 =edge( a, b, cap,0) ; edge e2=edge(b,a,0, 0) ;
g[a] . push_back ( ( int ) e. size ( ) ) ;
e. push_back ( e1 ) ;
    g [ b ] . push_back ( ( int ) e. size ( ) ) ;
      e. push_back ( e2 ) ;
}
bool bfs ( ) {
      int qh = 0 , qt = 0 ;
      q [ qt ++ ] = s ;
      memset ( d, -1 , sizeof d ) ;  d [ s ] = 0 ;
   while ( qh < qt && d [ t ] == - 1 ) {
     int v = q [ qh ++ ] ;
     for ( size_t i = 0 ; i < g [ v ] . size ( ) ; ++ i ) {
       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 ;
   }
int dinic ( ) {
      int flow = 0 ;
      for ( ;; ) {
if ( !bfs () )  break ;
         memset ( ptr, 0 , sizeof ptr ) ;
         while ( int pushed = dfs ( s, INF ) ) {
            flow += pushed ;
            if(pushed == 0)break;
         }
      }
      return flow ;
   }
```

MCMF:

```
const int MAXN = 7210;
const int MAXM = 100010;
const int INF = 1000000000;
struct Edge
{
   int u, v, cap, cost;
   int next;
         Edge(){}
         Edge(int u,int v,int cap,int cost,int
next):u(u),v(v),cap(cap),cost(cost),next(next){}
}edge[MAXM*3];
```

```
int NE;
int head[MAXN], dist[MAXN], pp[MAXN];
bool vis[MAXN];
void init(int n)
{
   NE = 0;
   fill(head,head+n+1,-1);
}
void addedge(int u, int v, int cap, int cost)
{
   edge[NE] = Edge(u,v,cap,cost,head[u]);
         head[u] = NE++;
   edge[NE] = Edge(v,u,0,-cost,head[v]);
   head[v] = NE++;
}
int qu[300010];
inline bool Min(int &x,int y)
{
   if(y < x) {x = y;return true;}
   return false;
}
bool SPFA(int s, int t, int n)
{
   int i, u, v , tmp;
   int he = 0 , ta = 0;
   fill(vis,vis+n+1,false);
   fill(pp,pp+n+1,-1);
   fill(dist,dist+n+1,INF);
   vis[s] = true; dist[s] = 0;
   qu[++ta] = s;
   while(he!=ta)
   {
      u = qu[++he];  vis[u] = false;
      if(he >= 250000) he = 0;
      for(i = head[u];~ i; i = edge[i].next)
      {
         v = edge[i].v;
                        tmp = dist[u] + edge[i].cost;
         if(edge[i].cap &&  dist[v] > tmp)
         {
            dist[v] = tmp;
            pp[v] = i;
            if(!vis[v])
            {
               qu[++ta] = v;
               if(ta >= 250000) ta = 0;
               vis[v] = true;
            }
         }
      }
   }
   if(dist[t] == INF) return false;
   return true;
}
pair<int,int> MCMF(int s, int t, int n) //
minCostMaxFlow
{
   int flow = 0; // 总流量
   int i, minflow, mincost;
   mincost = 0;
  for(;SPFA(s, t, n);)
   {
      minflow = INF + 1;
      for(i = pp[t]; ~i ; i = pp[edge[i].u])
            Min( minflow , edge[i].cap);
      flow += minflow;
      for(i = pp[t]; ~i; i = pp[edge[i].u])
      {
         edge[i].cap -= minflow;
         edge[i^1].cap += minflow;
      }
      mincost += dist[t] * minflow;
   }
   return make_pair(flow,mincost);
}
```

BPM:

```
int pre[201],vis[201];
vector<int>vc[201];
int find(int id) {
   int x,y,z;
   for(x=0;x<vc[id].size();x++) {
      if(vis[vc[id][x]]==0) {
         vis[vc[id][x]]=1;
         if(pre[vc[id][x]]==-1 || find(pre[vc[id][x]])) {
            pre[vc[id][x]]=id;
            return 1;    }   }      }
   return 0;
}
int bpm(int n) {
   int a=0,x;
   memset(pre,-1,sizeof pre);
   for(x=1;x<=n;x++) {
      memset(vis,0,sizeof vis);
      a+=find(x);
} return a;
}
```

STRING

KMP:

```
int k,l,c; // 1 indexed
char txt[1000001],pat[10000001];
int fnd[1000001],fn[1000001];
void comp()
{
   fn[1]=0;
   int m=0;
   for(int x=2;x<=l;x++)
   {
      while(m>0 && pat[m+1]!=pat[x])m=fn[m];
      if(pat[m+1]==pat[x])m++;
      fn[x]=m;
   }
}
void job()
{
   int b,d,e,f,x,y,z;
   b=0;
   for(x=1;x<=k;x++)
   {
      while(b>0 && pat[b+1]!=txt[x])b=fn[b];
      if(pat[b+1]==txt[x])b++;
      if(b==l)
      {
         c++;
         b=fn[b];
      }
   }
}
```

Finite Automata:

```
int TF[1001][27];
char pat[1001];
void computeFA(int M){
   int lps = 0;

   for(int i = 0; i < 26; i++) TF[0][i] = 0;
   TF[0][pat[0] - 'a'] = 1;

   for(int i = 1; i < M; i++){
      for(int j = 0; j < 26; j++) TF[i][j] = TF[lps][j];
      TF[i][pat[i] - 'a'] = i + 1;
      if(i < M) lps = TF[lps][pat[i] - 'a'];
   }
}
```

Manacher:

```
void pre_process(){
   s = "%";
   REP(i,tmp.size()){
      s += '#';
      s += tmp[i];
   }
   s += "#&";
}
LL P[300001];
void manacher(){    // manacher's algo - collected
from internet
   LL c = 0, r = 0;
   FOR(i, 1, s.size() - 1){
      LL i_mirror = 2 * c - i;
      P[i] = (r > i) ? min(r-i, P[i_mirror]) : 0;
      while (s[i + 1 + P[i]] == s[i - 1 - P[i]])P[i]++;
      if (i + P[i] > r) {
         c = i;
         r = i + P[i];
      }
   }
}
```

Z-Algo:

```
char s[1000003];
int zz[1000003];

void Z_algo(int 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++;
         zz[i] = R-L;
         R--;
      } else {
         int k = i-L;
         if (zz[k] < R-i+1) zz[i] = zz[k];
         else {
            L = i;
            while (R < n && s[R-L] == s[R]) R++;
            zz[i] = R-L;
            R--;
         }
      }
   }
}
```

Suffix Array:

```
int P[20][100001], N, stp, cnt;
char A[100001];

struct info {
   int nr[2], p;
   bool operator < ( const info &b ) const {
      return nr[0] == b.nr[0] ? ( nr[1] < b.nr[1] ) :
( nr[0] < b.nr[0] );
   }
} L[100001];

void suffixArray() {
   for( int i = 0; i < N; i++ ) P[0][i] = A[i] - 'a';
   for( stp = 1, cnt = 1; cnt >> 1 < N; ++stp, cnt <<=
1 ) {
      for( int i = 0; i < N; i++ ) {
         L[i].nr[0] = P[stp - 1][i];
         L[i].nr[1] = i + cnt < N ? P[stp - 1][i + cnt] :
-1;
         L[i].p = i;
      }
      sort(L, L + N);
      for ( int i = 0; i < N; i++ )
         P[stp][L[i].p] = i > 0 && L[i].nr[0] == L[i -
1].nr[0] && L[i].nr[1] == L[i - 1].nr[1] ? P[stp][L[i -
1].p] : i;
   }
}

int lcp(int x, int y) {
   int k, ret = 0;
   if(x == y) return N - x;
   for( k = stp - 1; k >= 0 && x < N && y < N; k-- )
if(P[k][x] == P[k][y])
      x += 1 << k, y += 1 << k, ret += 1 << k;
   return ret;
}
```

DATA STRUCTURE

LCA:

```
int pre[100001],lvl[100001];
int P[100001][22];
void pre_process(int N){
   memset(P,-1,sizeof P);
   for(int i = 1; i <= N; i++)P[i][0] = pre[i];
   for(int j = 1; (1<<j) <= N; j++){
      for(int i = 1; i <= N; i++){
      if(P[i][j - 1] != -1) P[i][j] = P[ P[i][j - 1] ][j - 1];
      }
   }
}
int LCA(int p, int q){
   if(lvl[p] < lvl[q])swap(p,q);
   int log;
   for(log = 1; (1<<log) <= lvl[p]; log++);
   log--;
   for(int i = log; i >= 0; i--){
      if(lvl[p] - (1<<i) >= lvl[q])p = P[p][i];
   }
   if(p == q)return p;
   for(int i = log; i >= 0; i--){
      if(P[p][i] != -1 && P[p][i] != P[q][i]){
         p = P[p][i];
         q = P[q][i];
      }
   }
   return pre[p];
}
```

DIVIDE AND CONQUER:

CONVEX HULL DP

```
int M[501][20001];
int B[501][20001];
int bad(int l1,int l2,int l3,int l){
   return (B[l][l3]-B[l][l1])*(M[l][l1]-M[l][l2])<(B[l]
[l2]-B[l][l1])*(M[l][l1]-M[l][l3]);
}
int cnt;
void insert_line(int m,int b,int l){
   M[l][cnt]=m;
   B[l][cnt++]=b;
   while (cnt>=3 && bad(cnt-3,cnt-2,cnt-1,l)>0){
      swap(M[l][cnt-1],M[l][cnt-2]);
      swap(B[l][cnt-1],B[l][cnt-2]);
      cnt--;
   }
}
int p;
int query(int x,int l)
{
   if(p>=cnt)p=cnt-1;
   while(p<cnt-1 && M[l][p+1]*x+B[l][p+1]<M[l]
[p]*x+B[l][p])p++;
   return M[l][p]*x+B[l][p];
}
```

SHANKS:

```
LL solve ( LL a, LL b, LL m ) {
	LL n = ( LL ) sqrt ( m + .0 ) + 1 ;

	LL an = 1 ;
	for ( int i = 0 ; i < n ; ++ i )
		an = ( an * a ) % m ;

	map < LL , LL > vals ;
	for ( int i = 1 , cur = an ; i <= n ; ++ i ) {
		if ( ! vals. count ( cur ) )
			vals [ cur ] = i ;
		cur = ( cur * an ) % m ;
	}
	LL ret = m;
	for ( int i = 0 , cur = b ; i <= n ; ++ i ) {
		if ( vals. count ( cur ) ) {
			int ans = vals [ cur ] * n - i ;
			if ( ans < ret )
				ret = ans;
		}
		cur = ( cur * a ) % m ;
	}
	if(ret < m)return ret;
	return - 1 ;
}
```

MATH:

Extended Euclid:

```
typedef long long i64;

int cases, caseno;

struct Euclid {
	i64 x, y, d;
	Euclid() {}
	Euclid( i64 xx, i64 yy, i64 dd ) { x = xx, y = yy, d = dd; }
};

Euclid egcd( i64 a, i64 b ) {
	if( !b ) return Euclid(1, 0, a);
	Euclid r = egcd ( b, a % b );
	return Euclid( r.y, r.x - a / b * r.y, r.d );
}

i64 abs64( i64 a ) {
	return a > 0 ? a : -a;
}

i64 myfloor( i64 a, i64 b ) {
	i64 c = a / b;
	if( (a % b) && a < 0 ) c--;
	return c;
}

i64 myceil( i64 a, i64 b ) {
	i64 c = a / b;
	if( (a % b) && a > 0 ) c++;
	return c;
}

i64 solve() {
	i64 a, b, c, x1, x2, y1, y2, X, Y, n1, n2, m1, m2;
	scanf("%lld %lld %lld %lld %lld %lld %lld", &a, &b, &c, &x1, &x2, &y1, &y2);

	c *= -1;
	if( a < 0 ) a *= -1, x1 *= -1, x2 *= -1, swap(x1, x2);
	if( b < 0 ) b *= -1, y1 *= -1, y2 *= -1, swap(y1, y2);

	if( !a && !b ) return !c ? (x2 - x1 + 1) * (y2 - y1 + 1) : 0;

	if( b == 0 ) {
		if( c % a ) return 0;
		i64 x = c / a;
		return (x >= x1 && x <= x2) * (y2 - y1 + 1);
	}
	if( a == 0 ) {
		if( c % b ) return 0;
		i64 y = c / b;
		return ( y >= y1 && y <= y2 ) * (x2 - x1 + 1);
	}

	Euclid s;

	s = egcd( a, b );

	if( c % s.d ) return 0;

	a /= s.d;
```

```
        b /= s.d;
        c /= s.d;
        s.d = 1;

        X = s.x * c;
        Y = s.y * c;

        n2 = min( myfloor( X - x1, b ), myfloor( y2 - Y, a ) );
        n1 = -min( myfloor( Y - y1, a ), myfloor( x2 - X, b ) );

        return (n2 < n1) ? 0 : n2 - n1 + 1;
}
```

Chinese Remainder Theorem:

```
i64 P[NN], R[NN];
int cases, caseno, n;

i64 bigmod( i64 a, i64 p, i64 MOD ) {
        i64 res = 1;
        while(p) {
                if( p & 1 ) res = (res * a) % MOD;
                a = (a * a) % MOD;
                p >>= 1;
        }
        return res;
}

int main() {
        freopen("a.in", "r", stdin);
        freopen("a.ans", "w", stdout);

        double cl = clock();

        scanf("%d", &cases);
        while( cases-- ) {
                scanf("%d", &n);
                for( int i = 0; i < n; i++ )
scanf("%lld %lld", &P[i], &R[i]);

                i64 lcm = 1, res = 0;
                for( int i = 0; i < n; i++ ) {
lcm *= P[i];
                        i64 rest = 1;
                        for( int j = 0; j < n; j++ ) if( i
!= j ) rest = rest * P[j];
res += bigmod( rest % P[i], P[i] - 2, P[i] ) * R[i]
* rest;
                }
                res %= lcm;
                printf("Case %d: %lld\n", +
+caseno, res);
}
```

Matrix Expo:

```
LL mul_res[MX][MX];
void multiply(LL a[][MX], LL b[][MX]){
   set0(mul_res);
   for(int i = 0; i < MX; i++){
      for(int j = 0; j < MX; j++){
         for(int k = 0; k < MX; k++)
            mul_res[i][j] = (mul_res[i][j] + a[i][k]
* b[k][j]) % mod;
      }
   }
}

LL big_res[MX][MX];
LL init[MX][MX];
void matrix_expo(LL n){
   if(n == 0){
      set0(big_res);
      REP(i, MX)big_res[i][i] = 1;
      return;
   }
   if(n % 2 == 0){
      matrix_expo(n / 2);
      multiply(big_res, big_res);
      REP(i, MX){
         REP(j, MX)big_res[i][j] = mul_res[i][j];
      }
   }
   else {
     matrix_expo(n - 1);
      multiply(big_res, init);
      REP(i, MX){
       REP(j, MX)big_res[i][j] = mul_res[i][j];
      }
   }
}
///       n <<= 1 ; /// not needed for this problem
```

FFT:

```
typedef complex < double > base ;

void fft ( vector < base > & a, bool invert ) {
        int n = ( int ) a. size ( ) ;

        for ( int i = 1 , j = 0 ; i < n ; ++ i ) {
                int bit = n >> 1 ;
                for ( ; j >= bit ; bit >>= 1 )
                        j -= bit ;
                j += bit ;
                if ( i < j )
                        swap ( a [ i ] , a [ j ] ) ;
        }

        for ( int len = 2 ; len <= n ; len <<= 1 ) {
                double ang = 2 * PI / len *
( invert ? - 1 : 1 ) ;
                base wlen ( cos ( ang ) , sin ( ang ) )
;
                for ( int i = 0 ; i < n ; i += len ) {
                        base w ( 1 ) ;
                        for ( int j = 0 ; j < len / 2 ; +
+ j ) {
                                base u = a [ i + j ] ,  v
= a [ i + j + len / 2 ] * w ;
                                a [ i + j ] = u + v ;
                                a [ i + j + len / 2 ] = u
- v ;
                                w *= wlen ;
                        }
                }
}
        if ( invert )
                for ( int i = 0 ; i < n ; ++ i )
                        a [ i ] /= n ;
}
int multiply ( const vector < int > & a, const
vector < int > & b) {
        vector < base > fa ( a. begin ( ) , a. end
( ) ) ,  fb ( b. begin ( ) , b. end ( ) ) ;
        size_t n = 1 ;
        while ( n < max ( a. size ( ) , b. size ( ) ) )
                n <<= 1 ;
        fa. resize ( n ) ,  fb. resize ( n ) ;

        fft ( fa, false ) ,  fft ( fb, false ) ;
        for ( size_t i = 0 ; i < n ; ++ i )
                fa [ i ] *= fb [ i ] ;
        fft ( fa, true ) ;
   int ret=0;
        for ( size_t i = 0 ; i < n ; ++ i )
   {
      int x=(int) ( fa [ i ] . real ( ) + 0.5 );
      if(x>0) ret++;
   }
   return ret;
}
```

CLOSEST PAIR:

```
const int N = int(1e5) + 10; //sorted by X axis
const LL INF = 1LL << 60;

struct Point {
    LL x,y;
} point[N];
int n;
int tmpt[N];

bool cmpxy(const Point& a, const Point& b) {
        if (a.x != b.x)
                return a.x < b.x;
        return a.y < b.y;
}

bool cmpy(const int& a, const int& b) {
        return point[a].y < point[b].y;
}

LL dis2(int i, int j) {
        return (point[i].x - point[j].x) *
(point[i].x - point[j].x)
                        + (point[i].y -
point[j].y) * (point[i].y - point[j].y);
}

LL sqr(LL x) {
        return x * x;
}
LL Closest_Pair(int left, int right) {
        LL d = INF;
        if (left == right)
                return d;
        if (left + 1 == right)
                return dis2(left, right);
        int mid = (left + right) >> 1;
        LL d1 = Closest_Pair(left, mid);
        LL d2 = Closest_Pair(mid + 1, right);
        d = min(d1, d2);
```

```
            int i, j, k = 0;
        for (i = left; i <= right; i++) {
                if (sqr(point[mid].x -
point[i].x) <= d)
                        tmpt[k++] = i;
        }
        sort(tmpt, tmpt + k, cmpy);
        for (i = 0; i < k; i++) {
                for (j = i + 1; j < k &&
sqr(point[tmpt[j]].y - point[tmpt[i]].y) < d;
                                j++) {
                        LL d3 =
dis2(tmpt[i], tmpt[j]);
                        if (d > d3)
                                d = d3;
                }
        }
        return d;
}
```

GAUSSIAN ELIMINATION:

```
int gauss()
{
   int x,y;
   for(x=0,y=0;x<row && y<col;y++)
   {
      int p=x;

      rep(i,x,row-1) if(a[i][y]) {p=i;break;} ///for
GF(2)
      rep(i,x+1,row-1) if(abs(a[i][y])>abs(a[p][y]))
p=i; ///for mod k
      rep(i,x+1,row-1) if(fabs(a[i][y])>fabs(a[p][y])
+eps) p=i;   ///for real values


      if(!a[p][y]) continue;  ///for GF(2) and mod k
      if(fabs(a[p][y])<eps) continue; ///for real values


      if(p!=x) rep(j,y,col) swap(a[p][j],a[x][j]);

      loc[y]=x; ///only for real value

      rep(i,0,row-1)
      {
         if(i==x || !a[i][y]) continue;  ///for GF(2) and
mod k
         if(i==x || fabs(a[i][y])<eps) continue; ///for
real values

         rep(j,y,col) a[i][j]^=a[x][j];  ///for GF(2)

         ll b=a[i][y],c=a[x][y]; ///for mod k
         rep(j,y,col) a[i][j] = ((c*a[i][j] - b*a[x][j])
%k+k)%k ; ///for mod k


         double tmp=a[i][y]/a[x][y]; ///for real values
         rep(j,y,col) a[i][j] -= a[x][j]*tmp; ///for real
values


      }
      x++;
   }

   /// for GF(2) and mod k
   int ans=1;

   rep(i,1,row-x) ans=(2LL*ans)%MOD;

   ans--;

   return (ans%MOD+MOD)%MOD;

   ///for real values only
   rep(j,0,col-1)
   {
      if(loc[j]==-1) continue;
      a[loc[j]][col]/=a[loc[j]][j];
      a[loc[j]][j]/=a[loc[j]][j];
   }


}

void init()
{
   num=row=col=100;
   CLR(a);
   SET(s);
   SET(loc);
}
int main() {
   init();
   gauss();
   printf("%.10lf\n",a[loc[pos]][col]);
}
```

GEOMETRY:

```
int dblcmp(double d)
{
   if (fabs(d)<eps)return 0;
   return d>eps?1:-1;
}
inline double sqr(double x){return x*x;}
/*
point()                                    - Empty constructor
point(double x, double y)      - constructor
input()                                    - double input
output()                                   - .2lf output
operator ==                                - compares x and y
operator <                                 - compares first by x, then by y
len()                                      - gives length from origin
len2()                                     - gives square of length from origin
distance(point p)                  - gives distance from p
add(point p)                       - returns new point after adding curresponging x and y
sub(point p)                       - returns new point after subtracting curresponging x and y
mul(double b)                      - returns new point after multiplieing x and y by b
div(double b)                      - returns new point after divideing x and y by b
dot(point p)                       - dot product
det(point p)                       - cross product of 2d points
rad(point a, point b)          - Probably radius of circumcircle of the triangle
trunc(double r)                    - return point that is truncated the distance from center to r
rotleft()                          - returns 90 degree ccw rotated point
rotright()                         - returns 90 degree cw rotated point
rotate(point p, double angle) - returns point after rotateing the point centering at p by angle radian ccw
*/
struct point
{
   double x,y;
   point()          {                          }
   point(double _x,double _y){    x = _x; y = _y;          }
   void input()       {   scanf("%lf%lf",&x,&y);          }
   void output()      {   printf("%.2f %.2f\n",x,y);       }
   bool operator==(point a)const{
      return dblcmp(a.x - x) == 0 && dblcmp(a.y - y) == 0;
   }
   bool operator<(point a)const{
      return dblcmp(a.x - x) == 0 ? dblcmp(y - a.y) < 0 : x < a.x;
   }
   point operator-(point a)const{
                  return point(x-a.x, y-a.y);
        }
   double len()       {   return hypot(x, y);             }
   double len2()      {   return x * x + y * y;           }
   double distance(point p){return hypot(x - p.x, y - p.y); }
   point add(point p)  {   return point(x + p.x, y + p.y);  }
```

```
    point sub(point p)  {   return point(x - p.x, y - p.y);  }
    point mul(double b) {   return point(x * b, y * b);      }
    point div(double b) {   return point(x / b, y / b);      }
    double dot(point p) {   return x*p.x+y*p.y;              }
    double det(point p) {   return x*p.y-y*p.x;              }
    double rad(point a,point b){
        point p=*this;
        return fabs(atan2(fabs(a.sub(p).det(b.sub(p))),a.sub(p).dot(b.sub(p))));
    }
    point trunc(double r){
        double l=len();
        if (!dblcmp(l))return *this;
        r/=l;
        return point(x*r,y*r);
    }
    point rotleft()     {   return point(-y,x);              }
    point rotright()    {   return point(y,-x);              }
    point rotate(point p,double angle){
        point v=this->sub(p);
        double c=cos(angle),s=sin(angle);
        return point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
    }
};
/*
Stores two points
line()                                          - Empty constructor
line(point a, point b)                  - line through a and b
operator ==                                     - checks if two points are same
line(point p, double angle)             - one end p, another end at angle degree
line(double a, double b, double c)      - line of equation ax + by + c = 0
input()                                         - inputs a and b
adjust()                                        - orders in such a way that a < b
length()                                        - distance of ab
angle()                                         - returns 0 <= angle < 180
relation()                                      - 0 if collinear
                                                        1 if ccw
                                                        2 if cw
pointonseg(point p)                     - returns 1 if point is on segment
parallel(line v)                        - returns 1 if they are parallel
segcrossseg(line v)                     - returns 0 if does not intersect
                                                        returns 1 if non-standard intersection
                                                        returns 2 if intersects
segcrossseg_inside(line v)              - returns 1 if intersects strictly inside
                                                        returns 0 if not
linecrossseg(line v)                    - v is line
linecrossline(line v)                   - 0 if parallel
                                                        1 if coincides
                                                        2 if intersects
crosspoint(line v)                      - returns intersection point
```

```
dispointtoline(point p)                    - distance from point p to the line
dispointtoseg(point p)                     - distance from p to the segment
lineprog(point p)                          - returns projected point p on ab line
symmetrypoint(point p)                     - returns reflection point of p over ab
*/
struct line
{
    point a,b;
    line()          {                       }
    line(point _a,point _b){ a=_a; b=_b;          }
    bool operator==(line v){ return (a==v.a)&&(b==v.b);     }
    line(point p,double angle){
        a=p;
        if (dblcmp(angle-pi/2)==0){
            b=a.add(point(0,1));
        }else{
            b=a.add(point(1,tan(angle)));
        }
    }
    //ax+by+c=0
    line(double _a,double _b,double _c){
        if (dblcmp(_a)==0){
            a=point(0,-_c/_b);
            b=point(1,-_c/_b);
        }else if (dblcmp(_b)==0){
            a=point(-_c/_a,0);
            b=point(-_c/_a,1);
        }else{
            a=point(0,-_c/_b);
            b=point(1,(-_c-_a)/_b);
        }
    }
    void input()       {   a.input(); b.input();          }
    void adjust()      {   if(b<a)swap(a,b);             }
    double length()    {   return a.distance(b);         }
    double angle(){
        double k=atan2(b.y-a.y,b.x-a.x);
        if (dblcmp(k)<0)k+=pi;
        if (dblcmp(k-pi)==0)k-=pi;
        return k;
    }
    int relation(point p){
        int c=dblcmp(p.sub(a).det(b.sub(a)));
        if (c<0)return 1;
        if (c>0)return 2;
        return 3;
    }
    bool pointonseg(point p){
        return dblcmp(p.sub(a).det(b.sub(a)))==0&&dblcmp(p.sub(a).dot(p.sub(b)))<=0;
```

```
}
bool parallel(line v){
    return dblcmp(b.sub(a).det(v.b.sub(v.a)))==0;
}
int segcrossseg(line v){
    int d1=dblcmp(b.sub(a).det(v.a.sub(a)));
    int d2=dblcmp(b.sub(a).det(v.b.sub(a)));
    int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));
    int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));
    if ((d1^d2)==-2&&(d3^d4)==-2)return 2;
    return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0||
                            d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0||
                            d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0||
                            d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);
}
int segcrossseg_inside(line v){
                if(v.pointonseg(a) || v.pointonseg(b) || pointonseg(v.a) || pointonseg(v.b)) return 0;
    int d1=dblcmp(b.sub(a).det(v.a.sub(a)));
    int d2=dblcmp(b.sub(a).det(v.b.sub(a)));
    int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));
    int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));
    if ((d1^d2)==-2&&(d3^d4)==-2)return 1;
    return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0||
                            d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0||
                            d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0||
                            d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);
}
int linecrossseg(line v){//*this seg v line
    int d1=dblcmp(b.sub(a).det(v.a.sub(a)));
    int d2=dblcmp(b.sub(a).det(v.b.sub(a)));
    if ((d1^d2)==-2)return 2;
    return (d1==0||d2==0);
}
int linecrossline(line v){
    if ((*this).parallel(v)){
        return v.relation(a)==3;
    }
    return 2;
}
point crosspoint(line v){
    double a1=v.b.sub(v.a).det(a.sub(v.a));
    double a2=v.b.sub(v.a).det(b.sub(v.a));
    return point((a.x*a2-b.x*a1)/(a2-a1),(a.y*a2-b.y*a1)/(a2-a1));
}
double dispointtoline(point p){
    return fabs(p.sub(a).det(b.sub(a)))/length();
}
double dispointtoseg(point p){
    if (dblcmp(p.sub(b).dot(a.sub(b)))<0||dblcmp(p.sub(a).dot(b.sub(a)))<0){
```

```
        return min(p.distance(a),p.distance(b));
      }
      return dispointtoline(p);
   }
   point lineprog(point p){
      return a.add(b.sub(a).mul(b.sub(a).dot(p.sub(a))/b.sub(a).len2()));
   }
   point symmetrypoint(point p){
      point q=lineprog(p);
      return point(2*q.x-p.x,2*q.y-p.y);
   }
};
```

```
/*  2D Geometry:  Circle tangents

================================================================
   Description: Given a circle (defined by a center point and radius)
                and a point strictly outside the circle, returns the
                two points of tangency.

   Complexity:  O(1)

----------------------------------------------------------------
   Author:      Ashley Zinyk
   Date:        Nov 19, 2002
   References:

----------------------------------------------------------------
   Reliability: 0
   Notes:       Assumes a non-zero distance between p and c.
*/

#define SQR(x) ((x)*(x))

typedef struct {
  double x, y;
} Point;

double dist2(Point a, Point b) {
  return SQR(a.x-b.x) + SQR(a.y-b.y);
}

Point a, b;
```

derangements : dr[i] = (i - 1) * (dr[i - 1] + dr[i - 2]);
dr[0] = dr[2] = 1; dr[1] = 0;

number of ways to ditribute n chocolate among r persons is (n+r-1)C(r-1)

picks theorem : A = i + (b/2) - 1 , i = interrior points, b = boundary points.

number of lattice points in between a segment of (x1, y1) and (x2,y2) is gcd(abs(x1-x2), abs(y1-y2)) + 1.

Strongly-connected components. Kosaraju's algorithm.
1. Let G T be a transpose G (graph with reversed edges.)
1. Call DFS(G T) to compute finishing times f [u] for each vertex u.
3. For each vertex u, in the order of decreasing f [u], perform DFS(G, u).
4. Each tree in the 3rd step's DFS forest is a separate SCC.
2-SAT. Build an implication graph with 2 vertices for each variable – for the variable and its inverse;
for each clause x ∨ y add edges (-x, y) and (-y, x).
The formula is satisfiable iff x and x are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

```
void circ_tangents(Point c, double r, Point p) {
  double perp, para, tmp = dist2(p,c);

  para = r*r/tmp;
  perp = r*sqrt(tmp-r*r)/tmp;

  a.x = c.x + (p.x-c.x)*para - (p.y-c.y)*perp;
  a.y = c.y + (p.y-c.y)*para + (p.x-c.x)*perp;
  b.x = c.x + (p.x-c.x)*para + (p.y-c.y)*perp;
  b.y = c.y + (p.y-c.y)*para - (p.x-c.x)*perp;
}


/* 2D-Geometry: Closest Point on a Line/Segment

================================================================================
   Description: Given the end points of a line segment, A and B, and
                another point C, returns the point on the segment
                closest to C.
                If a line perpendicular to A,B intersects A, B then
                intersection will be returned. Otherwise the closer
                endpoint will be returned.
                If the segment has a length of zero, an endpoint
                will be returned.

   Complexity:  O(1)

--------------------------------------------------------------------------------
   Author:      Ashley Zinyk
   Date:        Nov 9, 2002
   References:  0

--------------------------------------------------------------------------------
   Reliability: 1 (Spain 10263 - Railway)
   Notes:
*/
Point closest_pt_iline(Point a, Point b, Point c) {
  Point p;
  double dp;

  b.x -= a.x;
  b.y -= a.y;
  dp = (b.x*(c.x-a.x) + b.y*(c.y-a.y)) / (SQR(b.x)+SQR(b.y));
  p.x = b.x*dp + a.x;
  p.y = b.y*dp + a.y;
  return p;
}
Point closest_pt_lineseg(Point a, Point b, Point c) {
  Point p;
  double dp;

  b.x -= a.x;
  b.y -= a.y;
  if (fabs(b.x) < EPS && fabs(b.y) < EPS) return a;
  dp = (b.x*(c.x-a.x) + b.y*(c.y-a.y))/(SQR(b.x)+SQR(b.y));
  if (dp > 1) dp = 1;
  if (dp < 0) dp = 0;
  p.x = b.x*dp + a.x;
  p.y = b.y*dp + a.y;
  return p;
}

/* 2D-Geometry: Circle-Line Intersection

================================================================================
   Description: Given either a line or a line segment, and a circle
                these routines calculate the number and coordinates
                of the intersection points.

   Complexity:  O(1)

--------------------------------------------------------------------------------
   Author:      Gilbert Lee
   Date:        Sept 8, 2002
   References:  mathworld.wolfram.com/Circle-LineIntersection.html

--------------------------------------------------------------------------------
   Reliability: 0
```

```
*/
int sgn(double x){
  return x < 0 ? -1 : 1;
}

double dist_2d(Point a, Point b){
  return sqrt(SQR(a.x-b.x)+SQR(a.y-b.y));
}

int circ_iline_isect(Circle c, Point a, Point b,
                     Point *r1, Point *r2){
  double dx = b.x-a.x, dy = b.y-a.y;
  double sdr = SQR(dx)+SQR(dy), dr = sqrt(sdr);
  double D,disc,x,y;

  a.x -= c.o.x; a.y -= c.o.y;
  b.x -= c.o.x; b.y -= c.o.y;
  D = a.x*b.y - b.x*a.y;
  disc = SQR(c.r*dr)-SQR(D);

  if(disc < 0) return 0;
  x = sgn(dy)*dx*sqrt(disc);
  y = fabs(dy)*sqrt(disc);
  r1->x = (D*dy + x)/sdr + c.o.x;
  r2->x = (D*dy - x)/sdr + c.o.x;
  r1->y = (-D*dx + y)/sdr + c.o.y;
  r2->y = (-D*dx - y)/sdr + c.o.y;
  return disc == 0 ? 1 : 2;
}

int circ_lineseg_isect(Circle c, Point a, Point b,
                       Point *r1, Point *r2){
  double d = dist_2d(a,b);
  int res = circ_iline_isect(c,a,b,r1,r2);

  if(res == 2 && dist_2d(a,*r2)+dist_2d(*r2,b) != d)
res--;
  if(res >= 1 && dist_2d(a,*r1)+dist_2d(*r1,b) != d){
    *r1 = *r2;
    res--;
  }
  return res;
}

/* 2D Geometry: Minimum bounding circle

================================================
============================
```

```
  Notes:
  Description: Given a set of points, this returns the
circle with
               the minimum area which completely
contains all those
                    points

  Complexity:  O(n^3) worst case, where n is the
number of points
               but on average O(n)

----------------------------------------------------------
-
  Author:      Gilbert Lee
  Date:        Jan 24, 2003
  References:
http://www.cs.unc.edu/~eberly/gr_cont.htm

----------------------------------------------------------
-
  Reliability: 2 (Spain 10005 - Packing polygons)
               Problem C Aliens Jan 21, 2003
  Notes:       This is a simplification of the old
min_circle code
               by Scott Crosswhite (/Old/min_circle.c)
                    The input array is sorted to increase
stability of
                    an answer.  This may be removed to
increased speed
*/
int inside(Point p, Circle c){
  return SQR(p.x-c.x)+SQR(p.y-c.y) <= SQR(c.r);
}

Circle Circle1(Point p){
  Circle c;
  c.x = p.x; c.y = p.y; c.r = 0;
  return c;
}

Circle Circle2(Point p1, Point p2){
  Circle c;
  c.x = 0.5*(p1.x + p2.x);
  c.y = 0.5*(p1.y + p2.y);
  c.r = 0.5*sqrt(SQR(p1.x-p2.x)+SQR(p1.y-p2.y));
  return c;
}
```

```
  Circle res; double a,b,c,d,e,f,g;
  a =  p2.x - p1.x;  b = p2.y - p1.y;
  c =  p3.x - p1.x;  d = p3.y - p1.y;
  e = (p2.x + p1.x)*a + (p2.y + p1.y)*b;
  f = (p3.x + p1.x)*c + (p3.y + p1.y)*d;
  g = 2.0*(a*(p3.y - p2.y) - b*(p3.x - p2.x));
  if (fabs(g) < EPS){
    res.x = res.y = res.r = DBL_MAX;
    return res;
  }
  res.x = (d*e - b*f) / g;
  res.y = (a*f - c*e) / g;
  res.r = sqrt(SQR((p1.x-res.x))+SQR((p1.y-res.y)));
  return res;
}

Circle min_circle(Point *p, int n){
  int i, j, k; Point t; Circle c = Circle1(p[0]);

  /* Randomize point array to avoid doctored input -
may modify the
     limit on the for loop to increase/decrease
randomness */
  for(i = 0; i < n; i++){
    j = rand() % n;
    k = rand() % n;
    t = p[j]; p[j] = p[k]; p[k] = t;
  }

  for(i = 1; i < n; i++) if(!inside(p[i], c)){ c =
Circle1(p[i]);
  for(j = 0; j < i; j++) if(!inside(p[j], c)){ c =
Circle2(p[i],p[j]);
  for(k = 0; k < j; k++) if(!inside(p[k], c))  c =
Circle3(p[i],p[j],p[k]);}}
  return c;
}
bool pointInsideTriangle( double x[], double y[],
double xx, double yy )
{
   return leftTurn( x[0], y[0], x[1], y[1], xx, yy )
       && leftTurn( x[1], y[1], x[2], y[2], xx, yy )
       && leftTurn( x[2], y[2], x[0], y[0], xx, yy );
}
double polarAngle( P p )
{
   if( fabs( p.x ) <= EPS && fabs( p.y ) <= EPS )
return -1.0;
```

```
Circle Circle3(Point p1, Point p2, Point p3){
if( fabs( p.x ) <= EPS ) return ( p.y > EPS ? 1.0 :
3.0 ) * acos( 0 );
   double theta = atan( 1.0 * p.y / p.x );
   if( p.x > EPS ) return( p.y >= -EPS ? theta : ( 4 *
acos( 0 ) + theta ) );
   return( 2 * acos( 0 ) + theta );
}
bool pointInPoly( P p, vector< P > &poly )
{
   int n = poly.size();
   double ang = 0.0;
   for( int i = n - 1, j = 0; j < n; i = j++ )
   {
      P v( poly[i].x - p.x, poly[i].y - p.y );
      P w( poly[j].x - p.x, poly[j].y - p.y );
      double va = polarAngle( v );
      double wa = polarAngle( w );
      double xx = wa - va;
      if( va < -0.5 || wa < -0.5 || fabs( fabs( xx ) - 2 *
acos( 0 ) ) < EPS )
      {
         // POINT IS ON THE EDGE
         assert( false );
         ang += 2 * acos( 0 );
         continue;
      }
      if( xx < -2 * acos( 0 ) ) ang += xx + 4 *
acos( 0 );
      else if( xx > 2 * acos( 0 ) ) ang += xx - 4 * acos(
0 );
      else ang += xx;
   }
   return( ang * ang > 1.0 );
}
bool lineIntersect( P a, P b, P c, P d, P &r )
{
   P n; n.x = d.y - c.y; n.y = c.x - d.x;
   double denom = n.x * ( b.x - a.x ) + n.y * ( b.y -
a.y );
   if( fabs( denom ) < EPS ) return false;
   double num = n.x * ( a.x - c.x ) + n.y * ( a.y - c.y );
   double t = -num / denom;
   r.x = a.x + t * ( b.x - a.x );
   r.y = a.y + t * ( b.y - a.y );
   return true;
}
bool lineSegIntersect( vector< T > &x, vector< T >
```

```
   {
   double ucrossv1 = ( x[1] - x[0] ) * ( y[2] - y[0] ) - ( y[1] - y[0] ) * ( x[2] - x[0] );
   double ucrossv2 = ( x[1] - x[0] ) * ( y[3] - y[0] ) - ( y[1] - y[0] ) * ( x[3] - x[0] );
   if( ucrossv1 * ucrossv2 > 0 ) return false;
   double vcrossu1 = ( x[3] - x[2] ) * ( y[0] - y[2] ) - ( y[3] - y[2] ) * ( x[0] - x[2] );
   double vcrossu2 = ( x[3] - x[2] ) * ( y[1] - y[2] ) - ( y[3] - y[2] ) * ( x[1] - x[2] );
   return( vcrossu1 * vcrossu2 <= 0 );
}

/***************************
 * Circle through 3 points *
 ***************************
 * Computes the circle containing the 3 given points.
 * The 3 points are
 *      (x[0], y[0]), (x[1], y[1]) and (x[2], y[2]).
 * The centre of the circle is returned as (r[0], r[1]).
 * The radius is returned normally. If the circle is
 * undefined (the points are collinear), -1.0 is returned.
 * #include <math.h>
 * REQUIRES: lineIntersect
 * FIELD TESTING: Passed UVA 190
 **/
double circle3pts( double x[], double y[], double r[] )
{
   double lix[4], liy[4];
   lix[0] = 0.5 * ( x[0] + x[1] ); liy[0] = 0.5 * ( y[0] + y[1] );
   lix[1] = lix[0] + y[1] - y[0];  liy[1] = liy[0] + x[0] - x[1];
   lix[2] = 0.5 * ( x[1] + x[2] ); liy[2] = 0.5 * ( y[1] + y[2] );
   lix[3] = lix[2] + y[2] - y[1];  liy[3] = liy[2] + x[1] - x[2];
   if( !lineIntersect( lix, liy, r ) ) return -1.0;
   return sqrt(
      ( r[0] - x[0] ) * ( r[0] - x[0] ) +
      ( r[1] - y[0] ) * ( r[1] - y[0] ) );
}

/
*****************************************
 * Circle of a given radius through 2 points *
```

```
&y )
*****************************************
 * Computes the center of a circle containing the 2 given
 * points. The circle has the given radius. The returned
 * center is never to the right of the vector
 * (x1, y1)-->(x2, y2).
 * If this is possible, returns true and passes the center
 * through the ctr array. Otherwise, returns false.
 * #include <math.h>
 * FIELD TESTING:
 *      - Valladolid 10136: Chocolate Chip Cookies
 **/
bool circle2ptsRad( double x1, double y1, double x2, double y2, double r, double ctr[2] )
{
   double d2 = ( x1 - x2 ) * ( x1 - x2 ) + ( y1 - y2 ) * ( y1 - y2 );
   double det = r * r / d2 - 0.25;
   if( det < 0.0 ) return false;
   double h = sqrt( det );
   ctr[0] = ( x1 + x2 ) * 0.5 + ( y1 - y2 ) * h;
   ctr[1] = ( y1 + y2 ) * 0.5 + ( x2 - x1 ) * h;
   return true;
}
/****************
 * Great Circle *
 ****************
 * Given two pairs of (latitude, longitude), returns the
 * great circle distance between them.
 * FIELD TESTING
 *      - Valladolid 535: Globetrotter
 **/
double greatCircle( double laa, double loa, double lab, double lob )
{
   double PI = acos( -1.0 ), R = 6378.0;
   double u[3] = { cos( laa ) * sin( loa ), cos( laa ) * cos( loa ), sin( laa ) };
   double v[3] = { cos( lab ) * sin( lob ), cos( lab ) * cos( lob ), sin( lab ) };
   double dot = u[0]*v[0] + u[1]*v[1] + u[2]*v[2];
   bool flip = false;
   if( dot < 0.0 )
   {
```

```
        flip = true;
       for( int i = 0; i < 3; i++ ) v[i] = -v[i];
    }
    double cr[3] = { u[1]*v[2] - u[2]*v[1], u[2]*v[0] -
u[0]*v[2], u[0]*v[1] - u[1]*v[0] };
    double theta = asin( sqrt( cr[0]*cr[0] + cr[1]*cr[1]
+ cr[2]*cr[2] ) );
    double len = theta * R;
    if( flip ) len = PI * R - len;
    return len;
}
// 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;
}
bool cw(const point &a, const point &b, const point
&c) {
    return (b.first - a.first) * (c.second - a.second) -
(b.second - a.second) * (c.first - a.first) < 0;
}
vector<point> convexHull(vector<point> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    sort(p.begin(), p.end());
    int cnt = 0;
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[cnt++] = p[i++])
        for (; cnt >= 2 && !cw(q[cnt - 2], q[cnt - 1],
p[i]); --cnt)
            ;
    for (int i = n - 2, t = cnt; i >= 0; q[cnt++] = p[i--])
        for (; cnt > t && !cw(q[cnt - 2], q[cnt - 1], p[i]);
--cnt);
    q.resize(cnt - 1 - (q[0] == q[1]));
    return q;
}

    {
        int u,v;
```

```
struct trie
{
   int next[26],fail,score;
   void clear(){ rep(i,0,CHZ-1) next[i]=0;
fail=score=0; }
};

trie node[MX];

vi vv[MX];

int n,m,b,mp[300],val[30],sc[110];

char s[110][110];

void init()
{
   n=root+1;
   rep(i,0,1) node[i].clear();
   rep(i,0,CHZ-1) node[0].next[i]=root;
   rep(i,0,MX-1) vv[i].clear();

   return;
}

void insert(int x)
{
   int pos=root;
   for(int i=0;s[x][i];i++)
   {
      if(!node[pos].next[mp[s[x][i]]])
      {
         node[n].clear();
         node[pos].next[mp[s[x][i]]]=n++;
      }
      pos=node[pos].next[mp[s[x][i]]];
   }
   node[pos].score+=sc[x];
   return;
}

void get_failure()
{
   queue<int>q;

   q.push(root);

   while(!q.empty())
      u=q.front();
      q.pop();

      rep(i,0,CHZ-1)
      {
         if(node[u].next[i])
         {

node[node[u].next[i]].fail=node[node[u].fail].next[i];

node[node[u].next[i]].score+=node[node[node[u].nex
t[i]].fail].score;
            q.push(node[u].next[i]);
         }
         else {
            node[u].next[i]=node[node[u].fail].next[i];
         }
      }
   }
   return;
}

int dp[MX][205],vis[MX][205];

int go(int p,int c) {
   if(c > b) return -inf;
   int &ret=dp[p][c];
   if(vis[p][c] == kk) return ret;
   vis[p][c]=kk;
   ret=0;

   rep(i,0,CHZ-1) {
      if(c + val[i] <= b) {
         ret = max(ret, node[node[p].next[i]].score +
go(node[p].next[i],c+val[i]));
      }
   }

   return ret;
}
```

TEMPLATE:

```
#define PCASE printf("Case %d: ",kk++)
#define PCASENL printf("Case %d:\n",kk++)
#define NL puts("")
#define sz(a) ((int)a.size())
#define repv(i,a) for(int i=0;i<sz(a);i++)
#define revv(i,a) for(int i=sz(a)-1;i>=0;i--)
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define rev(i,a,b) for(int i=a;i>=b;i--)
#define FOR(I,A,B) for(int I = (A); I < (B); ++I)
#define REP(I,N)   FOR(I,0,N)
#define all(a) a.begin(),a.end()
#define pb(a) push_back(a)
#define mp(a,b) make_pair(a,b)
#define pi (2.0*acos(0.0))
#define PI (2.0*acos(0.0))
#define SET(a) memset(a,-1,sizeof a)
#define CLR(a) memset(a,0,sizeof a)
#define set0(ar)   memset(ar,0,sizeof ar)
#define setinf(ar) memset(ar,126,sizeof ar)
#define in(a,x,y) (a>=x && a<=y)
#define out(a,x,y) (!in(a,x,y))
#define xx first
#define yy second
using namespace std;

typedef long long ll;
typedef long long LL;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ll> vll;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<pii> vii;
template< class T > inline T _sq(T a) { return a * a; }
template< class T, class X > inline T _pow(T a,X y) {T z=1; rep(i,1,y){z*=a;} return z; }
template< class T > inline T _extended(T a,T b,T &x,T &y) {a=_abs(a);b=_abs(b); T g,x1,y1; if(!b)
{x=1;y=0;g=a; return g;} g=_extended(b,a%b,x1,y1); x=y1; y=x1-(a/b)*y1; return g;}
template< class T, class X > inline bool getbit(T a, X i) { T t=1; return ((a&(t<<i))>0); }
template< class T, class X > inline T setbit(T a, X i) { T t=1;return (a|(t<<i)); }
template< class T, class X > inline T resetbit(T a, X i) { T t=1;return (a&(~(t<<i))); }
template< class T, class X > inline T togglebit(T a, X i) { T t=1;return (a^(t<<i)); }

template< class T,class X, class Y > inline T _bigmod(T n,X m,Y mod){ull ret=1, a = n%mod ; while(m)
{ if(m&1)ret=(ret*a)%mod; m>>=1; a=(a*a)%mod; }ret%=mod;return (T)ret;}
template< class T, class Y > inline T _modinv(T n,Y mod) {return _bigmod(n,mod-2,mod);}
```