Table of Contents

Stable Marriage:

```
/* A person has an integer preference for each of the persons of the opposite
 * sex, produces a matching of each man to some woman. The matching will follow:
 *          - Each man is assigned to a different woman (n must be at least m)
 *          - No two couples M1W1 and M2W2 will be unstable.
 * Two couples are unstable if (M1 prefers W2 over W1 and W1 prefers M2 over M1)
 * INPUT: m - number of man, n - number of woman (must be at least as large as m)
 *          - L[i][]: the list of women in order of decreasing preference of man i
 *          - R[j][i]: the attractiveness of i to j.
 * OUTPUTS: - L2R[]: the mate of man i (always between 0 and n-1)
 *          - R2L[]: the mate of woman j (or -1 if single)      */
int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW], p[MAXM];
void stableMarriage() {
    memset( R2L, -1, sizeof( R2L ) );
    memset( p, 0, sizeof( p ) );
    for( int i = 0; i < m; i++ ) { // Each man proposes...
        int man = i;
        while( man >= 0 ) {
            int wom;
            while( 1 ) {
                wom = L[man][p[man]++];
                if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;
            }
            int hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby;
        }
    }
}
```

Stoer Wagner (Minimum Min Cut between All Pairs):

```
// Maximum edge weight (MAXW * NN * NN must fit into an int), NN number of vertices
#define MAXW 1000
int g[NN][NN], v[NN], w[NN], na[NN]; // Adjacency matrix and some internal arrays
bool a[NN];
int minCut( int n ) { // 0 indexed
    int i, j;
    for( i = 0; i < n; i++ ) v[i] = i; // init the remaining vertex set
    int best = MAXW * n * n;
    while( n > 1 ) { // initialize the set A and vertex weights
        a[v[0]] = true;
        for( i = 1; i < n; i++ ) {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }
        int prev = v[0];
        for(i = 1; i < n; i++) { // find the most tightly connected non-A vertex
            int zj = -1;
            for(j=1;j<n;j++) if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) ) zj = j;
            a[v[zj]] = true; // add it to A
            if( i == n - 1 ) { // last vertex?
                best = (best < w[zj]) ? best : w[zj]; // remember the cut weight
                for(j=0;j<n;j++) g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
            for( j = 1; j < n; j++ ) if( !a[v[j]] ) w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}
```

Euler's Formula:

If G is a connected plane graph with v vertices, e edges, and f faces, then
v - e + f = 1 + number of connected components.

MST (Directed Graph):

1. For each node (except the root), look for the minimum weight incoming edge.
2. Look for cycles, if there's no cycle, we already have a tree (which is an MST) goto End
3. Pick one cycle and find an edge p->q, p is in set (not part of the cycle). q is in set (s part of the cycle). Pick this p and q such that: cost of (p->q + sum of all edges in the cycle) - the minimum incoming edge to q (computed in step 1) is minimum. Return to step 2.

```
struct edge { // Caution: The vertices should be reachable from the root
    int v, w;
    bool operator < ( const edge &v ) const { return w > v.w; }
};
vector <edge > adj[MAX];    // For saving incoming edges and their costs
int DMST( int n, int root ) {   // 1 indexed
    int i, res=0, pr[MAX], cost[MAX], sub[MAX], sn[MAX], visited[MAX];
    vector <int> ::iterator v, it;
    vector <int> node[MAX];
    for(i = 0; i <= n; i++) {
        node[i].clear(); node[i].push_back( i );
        sn[i] = i, sub[i] = pr[i] = 0;
    }
    for(i = 1; i <= n; i++) if( i != root ) {
        sort( adj[i].begin(), adj[i].end() ); // sorted in descending order of w
        pr[i] = adj[i].back().v;
        cost[i] = sub[i] = adj[i].back().w;
        res += cost[i];
    }
    bool cycle = true;
    while( cycle ) {
        cycle = false;
        memset( visited, 0, sizeof( visited ) );
        for(i = 1; i <= n; i++) {
            if( visited[i] || sn[i] != i ) continue;
            int cur = i;
            do {
                visited[cur] = i;
                cur = pr[cur];
            }while( cur > 0 && !visited[ cur ] );
            if( cur > 0 && visited[ cur ] == i ) {
                cycle = true;
                int start = cur; // assert( sn[start] == start ) ;
                do{
                    if( *node[cur].begin() != cur ) break;
                    for( it = node[cur].begin(); it != node[cur].end(); it++) {
                        sn[ *it ] = start;
                        if( cur != start ) node[ start ].push_back ( *it );
                    }
                    if( cur != start ) node[ cur ].clear();
                    cur = pr[ cur ];
                }while( cur != start );
                int best = INT_MAX;
                for( v = node[start].begin(); v!=node[start].end(); v++) {
                    while( !adj[*v].empty() && sn[adj[*v].back().v] == start)
                        adj[ *v ].pop_back();
                    if( !adj[*v].empty() ) {
                        int tcost = adj[*v].back().w - sub[ *v ];
                        if( tcost < best ) best = tcost, pr[ start ] = adj[*v].back().v;
                    }
                } //assert( best >= 0 && best < INT_MAX );
                cost[ start ] = best;
                for( v = node[start].begin(); v != node[start].end(); v++ )sub[*v] += best;
                res += best;
            }
        }
        for(i = 1; i <= n; i++) pr[i] = sn[ pr[i] ];
    }
    return res;
}
```

Bipartite Matching:

```
int adj[MAX][MAX], deg[MAX], Left[MAX], Right[MAX], m, n;
bool visited[MAX];
bool bpm( int u ) {
    for(int i = 0, v; i < deg[u]; i++) {
        v = adj[u][i];
        if( visited[v] ) continue;
        visited[v] = true;
        if( Right[v] == -1 || bpm( Right[v] ) ) {
            Right[v] = u, Left[u] = v;
            return true;
        }
    }
    return false;
}
int bipartiteMatching() { // Returns Maximum Matching
    memset ( Left, -1, sizeof( Left ) );
    memset ( Right, -1, sizeof( Right ) );
    int i, cnt=0;
    for( i = 0; i < m; i++ ) {
        memset( visited, 0, sizeof( visited ) );
        if( bpm( i ) ) cnt++;
    }
    return cnt;
}
```

Erdos and Gallai Theorem:

```
// Given the degrees of the vertices of a graph, is it possible to construct such graph
Input - the deg[] array
int deg[MM], n, degSum[MM], ind[MM], minVal[MM];
bool ErdosGallai() { // 1 indexed
    bool poss = true;
    int i, sum = 0, j, r;
    for( i = 1; i <= n; i++ ) {
        if( deg[i] >= n ) poss = false;
        sum += deg[i];
    }
    //Summation of degrees has to be ODD and all degrees has to be < n - 1
    if( !poss || ( sum & 1 ) || ( n == 1 && deg[1] > 0 ) ) return false;
    sort( deg + 1, deg + n + 1, greater <int>() );
    degSum[0] = 0;
    j = n;
    for( i = 1; i <= n; i++ ) {
        degSum[i] = degSum[i-1] + deg[i];    //CONSTRUCTING: degSum
        for( ; j >= 1 && deg[j] < i; j-- );  //CONSTRUCTING: ind
        ind[i] = j+1;
    }
    //CONSTRUCTING : minVal
    for(r = 1; r < n; r++) {
        j = ind[r];
        if( j == n+1 ) minVal[r]=( n - r ) * r;
        else if( j <= r ) minVal[r] = degSum[n] - degSum[r];
        else {
            minVal[r] = degSum[n] - degSum[j-1];
            minVal[r] += (j-r-1)*r;
        }
    }
    //Checking : Erdos & Gallai Theorem
    for( r = 1; r < n; r++ ) if( degSum[r] > ( r * (r-1) + minVal[r] ) ) return false;
    return true;
}
```

Catalan Number:

$$C_n = \frac{2n!}{n!(n+1)!} \qquad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

Maximum Flow (Dinic) with Min Cut:

```
// cap[][] and Cap[][] both contains the capacity, cap reduces after the flow
int deg[MAX], adj[MAX][MAX], cap[MAX][MAX], Cap[MAX][MAX], q[100000];
int dinic( int n,int s,int t ) {
    int prev[MAX], u, v, i, z, flow = 0, qh, qt, inc;
    while(1) {
        memset( prev, -1, sizeof( prev ) );
        qh = qt = 0;
        prev[s] = -2;
        q[qt++] = s;
        while( qt != qh && prev[t] == -1 ) {
            u = q[qh++];
            for(i = 0; i < deg[u]; i++) {
                v = adj[u][i];
                if( prev[v] == -1 && cap[u][v] ) {
                    prev[v] = u;
                    q[qt++] = v;
                }
            }
        }
        if(prev[t] == -1) break;
        for(z = 0; z < n; z++) if( prev[z] !=- 1 && cap[z][t] ) {
            inc = cap[z][t];
            for( v = z, u = prev[v]; u >= 0; v = u, u=prev[v]) inc = min( inc, cap[u][v] );
            if( !inc ) continue;
            cap[z][t] -= inc;
            cap[t][z] += inc;
            for(v=z, u = prev[v]; u >= 0; v = u, u = prev[v]) {
                cap[u][v] -= inc;
                cap[v][u] += inc;
            }
            flow += inc;
        }
    }
    return flow;
}
bool visited[MAX];
void dfs( int u ) {
    visited[u] = true;
    for(int i = 0; i < deg[u]; i++) {
        int v = adj[u][i];
        if( !visited[v] && cap[u][v] && Cap[u][v] ) dfs(v);
    }
}
void printMincut( int s ) {
    memset( visited, 0, sizeof( visited ) );
    dfs( s );
    for(int u = 0; u < n; u++) if( visited[u] )
        for(int v = 0; v < n; v++) if( !visited[v] && !cap[u][v] && Cap[u][v] )
            printf("%d %d\n", u+1, v+1);
}
```

Extended Euclid & GCD:

```
struct Euclid {
    int x, y, d;
    Euclid() {}
    Euclid( int xx, int yy, int dd ) { x = xx, y = yy, d = dd; }
};
int gcd( int a, int b ) { return !b ? a : gcd ( b, a % b ); }
Euclid egcd( int a, int b ) { // Input a, b; Output x, y, d; ax + by = d, d = gcd(a,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 );
}
```

Articulation Point:

```
// result[i] will contain true if the ith node is an articulation
int adj[MAX][MAX], deg[MAX], n, visited[MAX], assignedVal;
bool result[MAX];
int articulation( int u, int depth ) {
    if( visited[u] > 0 ) return visited[u];
    visited[u] = ++assignedVal;
    int mn = visited[u], rootCalled = 0;
    for(int i = 0; i < deg[u]; i++) {
        int v = adj[u][i];
        if( !visited[v] ) {
            if( !depth ) rootCalled++;
            int k = articulation( v, depth + 1 );
            if( k >= visited[u] ) result[u] = true;
        }
        mn = min( mn, visited[v] );
    }
    if( !depth ) result[u] = ( rootCalled >= 2 );
    return visited[u] = mn;
}
void processArticulation() {
    assignedVal = 0;
    memset( result, 0, sizeof( result ) );
    memset( visited, 0, sizeof( visited ) );
    for( int i = 0; i < n; i++ ) if( !visited[i] ) articulation( i, 0 );
}
```

Bridge Detection:

```
// prints the bridges in arbitrary order
int adj[MAX][MAX], deg[MAX], n, visited[MAX], assignedVal;
int bridge( int u, int parent ) {
    visited[u] = ++assignedVal;
    int mn = visited[u];
    for( int i = 0; i < deg[u]; i++ ) if( adj[u][i] != parent ) {
        int v = adj[u][i];
        if( !visited[v] ) {
            int k = bridge( v, u );
            if( k > visited[u] ) printf("%d - %d\n", u, v);
        }
        mn = min( mn, visited[v] );
    }
    return visited[u] = mn;
}
void processBridge() {
    assignedVal = 0;
    memset( visited, 0, sizeof( visited ) );
    for( int i = 0; i < n; i++ ) if( !visited[i] ) bridge( i, -1 );
}
```

Euler Phi Function (Sieve Version):

```
int Phi[MAX];
void sievePHI() { // Phi[i] = phi(i), uses the idea of sieve
    Phi[1] = 1;
    int i, j;
    for( i = 2; i < MAX; i++ ) if( !Phi[i] ) {
        Phi[i] = i - 1;
        for( j = i + i; j < MAX; j += i ) {
            if( !Phi[j] ) Phi[j] = j;
            Phi[j] = Phi[j] / i * ( i - 1 );
        }
    }
}
```

Pick's Theorem:

```
// Only for integer points
I = area + 1 - B/2
Where   I = number of points inside
        B = number of points on the border
```

Euler Phi Function:

```
int phi( int n ) { // Uses Prime Factoring of n
    int factors[NN], factorCount[NN], factorLen;
    findPrimeFactors( n, factors, factorCount, factorLen );
    for( int i = 0; i < factorLen; i++ ) {
        n /= factors[i];
        n *= factors[i] - 1;
    }
    return n;
}
```

Prime Factoring of n:

```
// factors[]-contains all factors, factorCount[]-contains frequency, factorLen-length
void findPrimeFactors( int n, int *factors, int *factorCount, int &factorLen ) {
    int i, cnt, sqrtN = ( int ) sqrt ( ( double ) n ) + 1;
    factorLen = 0;
    for( i = 0; pr[i] < sqrtN; i++ ) if( !( n % pr[i] ) ) {
        factors[factorLen] = pr[i], cnt = 0;
        while( !( n % pr[i] ) ) n /= pr[i], cnt++;
        factorCount[factorLen++] = cnt;
        sqrtN = ( int ) sqrt ( ( double ) n ) + 1;
    }
    if( n > 1 ) factors[factorLen] = n, factorCount[factorLen++] = 1;
}
```

Sieve for Finding Primes:

```
// prime upto - PrimeLIMIT, pr[] contains the primes, prlen-length of pr[]
int prime[PrimeLIMIT / 64], pr[MAX_TOTAL], prlen;
#define gP(n) (prime[n>>6]&(1<<((n>>1)&31)))
#define rP(n) (prime[n>>6]&=~(1<<((n>>1)&31)))
void sieve() {
    unsigned int i,j,sqrtN,i2;
    memset( prime, -1, sizeof( prime ) );
    sqrtN = ( int ) sqrt ( ( double ) PrimeLIMIT ) + 1;
    for( i = 3; i < sqrtN; i += 2 ) if( gP( i ) )
        for( j = i * i, i2 = i << 1; j < PrimeLIMIT; j += i2 ) rP( j );
    pr[prlen++] = 2;
    for( i = 3; i < PrimeLIMIT; i += 2 ) if( gP( i ) ) pr[prlen++] = i;
}
```

Divisors of n:

```
// *divisor-divisors list(not sorted), divisorLen-*divisor length, MM should be bigger
void findDivisors( int n, int *divisors, int &divisorLen ) {
    int factors[NN], factorCount[NN], factorLen, st[MM][2], top = 0;
    findPrimeFactors( n, factors, factorCount, factorLen ); // Prime Factoring of n
    divisorLen = 0;
    int result, i, j, k;
    st[top][0] = 1, st[top++][1] = 0;
    while( top-- ) {
        result = st[top][0];
        i = st[top][1];
        if( i == factorLen ) {
            divisors[divisorLen++] = result;
            continue;
        }
        for( j = 0, k = 1; j <= factorCount[i]; j++, k *= factors[i] )
            st[top][0] = result * k, st[top++][1] = i + 1;
    }
}
```

Modular Inverse:

```
int modularInverse( int a, int n ) { // given a and n, returns x, ax mod n = 1
    Euclid t = egcd( a, n );
    if( t.d > 1 ) return 0;
    int r = t.x % n;
    return r < 0 ? r + n : r;
}
```

Modular Linear Equation Solver:

```
// Input - a, b, n; Output - all x in a vector; ax = b (mod n)
vector <int> modularEqnSolver( int a, int b, int n ) {
    Euclid t = egcd( a, n );
    vector <int> r;
    if( b % t.d ) return r;
    int x = ( b / t.d * t.x ) % n;
    if( x < 0 ) x += n;
    for( int i = 0; i < t.d; i++ ) r.push_back( ( x + i * n / t.d ) % n );
    return r;
}
```

Prime Factoring of n!:

```
// factors[]-contains all factors, factorCount[]-contains frequency, factorLen-length
void findPrimeFactorsOfFactorial(int n,int *factors,int *factorCount,int &factorLen) {
    factorLen = 0;
    for( int i = 0; pr[i] <= n; i++ ) {
        int cnt = 0;
        for( int j = pr[i]; j <= n; j *= pr[i] ) cnt += n / j;
        factors[factorLen] = pr[i], factorCount[factorLen++] = cnt;
    }
}
```

Kth Best Shortest Path:

```
int m, n, deg[MM], source, sink, K, val[MM][12];
struct edge {
    int v, w;
}adj[MM][500];
struct info {
    int v, w, k;
    bool operator < ( const info &b ) const {
        return w > b.w;
    }
};
priority_queue < info, vector <info> > Q;
void kthBestShortestPath() {
    int i, j;
    info u, v;
    for( i = 0; i < n; i++ ) for( j = 0; j < K; j++ ) val[i][j] = inf;
    u.v = source; u.k = 0; u.w = 0;
    Q.push(u);
    while( !Q.empty() ) {
        u = Q.top();
        Q.pop();
        for( i = 0; i < deg[u.v]; i++ ) {
            v.v = adj[u.v][i].v;
            int cost = adj[u.v][i].w + u.w;
            for( v.k = u.k; v.k < K; v.k++ ) {
                if( cost == inf ) break;
                if( val[v.v][v.k] > cost ) {
                    swap( cost, val[v.v][v.k] );
                    v.w = val[v.v][v.k];
                    Q.push(v);
                    break;
                }
            }
            for( v.k++; v.k < K; v.k++ ) {
                if( cost == inf ) break;
                if( val[v.v][v.k] > cost ) swap( cost, val[v.v][v.k] );
            }
        }
    }
}
```

Min Cost Max Flow Optimized (Dijkstra + Johnson):

```
// Input-**cap, **cost; Output-fcost, flow in fnet, fnet[u][v]-fnet[v][u] is net flow
int cap[NN][NN],fnet[NN][NN],adj[NN][NN],deg[NN],pr[NN],d[NN],pi[NN],cost[NN][NN];
#define Pot(u,v) (d[u] + pi[u] - pi[v])
bool dijkstra( int n, int s, int t ) {
    int i;
    for( i = 0; i < n; i++ ) d[i] = inf, pr[i] = -1;
    d[s] = 0;
    pr[s] = -n - 1;
    while( 1 ) {
        int u = -1, bestD = inf;
        for( i = 0; i < n; i++ ) if( pr[i] < 0 && d[i] < bestD ) bestD = d[u = i];
        if( bestD == inf ) break;
        pr[u] = -pr[u] - 1;
        for( i = 0; i < deg[u]; i++ ) {
            int v = adj[u][i];
            if( pr[v] >= 0 ) continue;
            if( fnet[v][u] && d[v] > Pot(u,v) - cost[v][u] )
                d[v] = Pot( u, v ) - cost[v][u], pr[v] = -u - 1;
            if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )
                d[v] = Pot(u,v) + cost[u][v], pr[v] = -u - 1;
        }
    }
    for( i = 0; i < n; i++ ) if( pi[i] < inf ) pi[i] += d[i];
    return pr[t] >= 0;
}
int mcmf3( int n, int s, int t, int &fcost ) {
    int u, v, flow = 0;
    fcost = 0;
    CLR( deg );
    for( u = 0; u < n; u++ ) for( v = 0; v < n; v++ ) if( cap[u][v] || cap[v][u] )
        adj[u][ deg[u]++ ] = v;
    CLR( fnet );
    CLR( pi );
    while( dijkstra( n, s, t ) ) {
        int bot = INT_MAX;
        for( v = t, u = pr[v]; v != s; u = pr[v = u] )
            bot = min ( bot, fnet[v][u] ? fnet[v][u] : ( cap[u][v] - fnet[u][v] ) );
        for( v = t, u = pr[v]; v != s; u = pr[v = u] )
            if( fnet[v][u] ) { fnet[v][u] -= bot; fcost -= bot * cost[v][u]; }
            else { fnet[u][v] += bot; fcost += bot * cost[u][v]; }
        flow += bot;
    }
    return flow;
}
```

KMP Matcher(T, P)

```
1    n = length(T)
2    m = length(P)
3    pi = ComputePrefixFunction(P)
4    q = 0
5    for i = 1 to n
6       while q > 0 and P[q+1] != T[i] do q = pi[q]
7       if P[q+1] == T[i] then q = q + 1
8       if q == m then print "Pattern occurs with shift i-m"
9          q = pi[q]                    //Look for the next match
```

ComputePrefixFunction(P)

```
1    m = length(P)
2    pi[1] = k = 0
3    for q = 2 to m
4       while k > 0 and P[k+1] != P[q] do k = pi[k]
5       if P[k+1] == P[q] then k = k + 1
6       pi[q] = k
7    return pi
```

Min Cost Max Flow (Bellman Ford):

```
// Input-**cap, **cost; Output-fcost, reduces cap, Works only for directed graphs
// The back edge cost will be the negative of the forward edge cost
int cap[NN][NN], pr[NN], cost[NN][NN], d[NN], adj[NN][NN], deg[NN];
bool bellmanford( int n, int s, int t ) {
    for( int i = 0; i < n; i++ ) d[i] = inf;
    d[s] = 0, pr[s] = -1;
    bool flag = true;
    for( int it = 0; it < n - 1 && flag; it++ ) {
        flag = false;
        for( int u = 0; u < n; u++ )
            for( int i = 0; i < deg[u]; i++ )s {
                int v = adj[u][i];
                if( cap[u][v] && d[v] > d[u] + cost[u][v] )
                    d[v] = d[u] + cost[u][v], pr[v] = u, flag = true;
            }
    }
    return d[t] < inf;
}
int mcmf2( int n, int s, int t, int &fcost ) {
    int netFlow = 0, u, v;
    fcost = 0;
    CLR( deg );
    for( u = 0; u < n; u++ ) for( v = 0; v < n; v++ ) if( cap[u][v] || cap[v][u] )
        adj[u][ deg[u]++ ] = v;
    while( bellmanford( n, s, t ) ) {
        int bot = inf;
        for( v = t; pr[v] != -1; v = pr[v] ) bot = min( bot, cap[ pr[v] ][v] );
        for( v = t; pr[v] != -1; v = pr[v] ) {
            cap[ pr[v] ][v] -= bot;
            cap[v][ pr[v] ] += bot;
            fcost += bot * cost[ pr[v] ][v];
        }
        netFlow += bot;
    }
    return netFlow;
}
```

Least Common Ancestor (LCA):

```
// N is the number of nodes, T[i] contains the parent of i, calculates P[][]
// First cal preprocessLCA, then run the queryLCA for each query
void preprocessLCA( int N, int T[MAXN], int P[MAXN][LOGMAXN] ) { // 0 indexed
    int i, j;
    //we initialize every element in P with -1
    for( i = 0; i < N; i++ ) for( j = 0; 1 << j < N; j++ ) P[i][j] = -1;
    for( i = 0; i < N; i++ ) P[i][0] = T[i]; //the first ancestor of i is T[i]
    for( j = 1; 1 << j < N; j++ ) for( i = 0; i < N; i++ ) //bottom up dp
        if( P[i][j - 1] != -1 ) P[i][j] = P[P[i][j - 1]][j - 1];
}
// L is the depth/level array, should be precalculated
int queryLCA( int N, int P[MAXN][LOGMAXN], int T[MAXN], int L[MAXN], int p, int q ) {
    int tmp, log, i;
    //if p is situated on a higher level than q then we swap them
    if( L[p] < L[q] ) tmp = p, p = q, q = tmp;
    for( log = 1; 1 << log <= L[p]; log++ ); //we compute the value of [log(L[p)]
    log--;
    //we find the ancestor of p situated on the same level with q using the values in P
    for( i = log; i >= 0; i-- ) if( L[p] - (1 << i) >= L[q] ) p = P[p][i];
    if( p == q ) return p;
    //we compute LCA(p, q) using the values in P
    for( 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 T[p];
}
```

Weighted Bipartite Matching O(n^3):

```
// Take input in cost[][]
#define N 55
#define INF 100000000

int cost[N][N], n, max_match;
int lx[N], ly[N];
int xy[N], yx[N];
bool S[N], T[N];
int slack[N], slackx[N], prev[N];

void init_labels() {
    memset( lx, 0, sizeof(lx) );
    memset( ly, 0, sizeof(ly) );
    for( int x = 0; x < n; x++ ) for( int y = 0; y < n; y++ ) lx[x] = max(lx[x],
cost[x][y]);
}
void update_labels() {
    int x, y, delta = INF;
    for (y = 0; y < n; y++) if (!T[y]) delta = min(delta, slack[y]);
    for (x = 0; x < n; x++) if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++) if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++) if (!T[y]) slack[y] -= delta;
}
void add_to_tree( int x, int prevx ) {
    S[x] = true;
    prev[x] = prevx;
    for (int y = 0; y < n; y++)
        if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
            slack[y] = lx[x] + ly[y] - cost[x][y];
            slackx[y] = x;
        }
}
void augment() {
    if( max_match == n ) return;
    int x, y, root;
    int q[N], wr = 0, rd = 0;
    memset(S, false, sizeof(S));
    memset(T, false, sizeof(T));
    memset(prev, -1, sizeof(prev));
    for( x = 0; x < n; x++ ) if (xy[x] == -1) {
        q[wr++] = root = x;
        prev[x] = -2;
        S[x] = true;
        break;
    }
    for( y = 0; y < n; y++ ) {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }
    while( true ) {
        while( rd < wr ) {
            x = q[rd++];
            for( y = 0; y < n; y++ )
                if( cost[x][y] == lx[x] + ly[y] && !T[y] ) {
                    if( yx[y] == -1 ) break;
                    T[y] = true;
                    q[wr++] = yx[y];
                    add_to_tree( yx[y], x);
                }
            if(y < n) break;
        }
        if(y < n) break;
        update_labels();
        wr = rd = 0;
```

```
        for(y = 0; y < n; y++) if(!T[y] &&  slack[y] == 0) {
            if(yx[y] == -1) {
                x = slackx[y];
                break;
            }
            else {
                T[y] = true;
                if (!S[yx[y]]) {
                    q[wr++] = yx[y];
                    add_to_tree(yx[y], slackx[y]);
                }
            }
        }
        if(y < n) break;
    }
    if(y < n) {
        max_match++;
        for( int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty ) {
            ty = xy[cx];
            yx[cy] = cx;
            xy[cx] = cy;
        }
        augment();
    }
}
int hungarian() {
    int ret = 0;
    max_match = 0;
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels();
    augment();
    for(int x = 0; x < n; x++) ret += cost[x][xy[x]];
    return ret;
}
```

Gaussian Elimination (float) :

```
void gauss( int N, long double mat[NN][NN] ) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        k = i;
        for (j = i+1; j < N; j++) if (fabs(mat[j][i]) > fabs(mat[k][i])) k = j;
        if (k != i) for (j = 0; j <= N; j++) swap(mat[k][j], mat[i][j]);
        for (j = i+1; j <= N; j++) mat[i][j] /= mat[i][i];
        mat[i][i] = 1;
        for (k = 0; k < N; k++) if( k != i ) {
            long double t = mat[k][i];
            if (t == 0.0L) continue;
            for (j = i; j <= N; j++) mat[k][j] -= t * mat[i][j];
            mat[k][i] = 0.0L;
        }
    }
}
```

Segmented Sieve:

```
void segmented_sieve( int l , int h ) {
    int i , j , k , m , end ;
    double L = (double) l ;
    memset ( composite , 0 , sizeof ( composite ) ) ;
    end = ceil ( sqrt ( h ) ) ;
    for ( i=3 ; i<end ; i+=2 ) {
        if ( !COMPS[i] ){
            j = ceil ( L / i ) ;    k = h / i ; m = i*j-l ;
            for ( j , m ; j<=k ; j++ , m+=i )   composite[m] = 1 ;
        }
    }
}
```

Fitting A Rectangle Inside Another Rectangle:

```
// Checks whether ractangle with sides (a, b) fits into rectangle with sides (c, d)
bool fits( int a, int b, int c, int d ) {
    double X, Y, L, K, DMax;
    if( a < b ) swap( a, b );
    if( c < d ) swap( c, d );
    if( c <= a && d <= b ) return true;
    if( d >= b ) return false;
    X = sqrt( a*a + b*b );
    Y = sqrt( c*c + d*d );
    if( Y < b ) return true;
    if( Y > X ) return false;
    L = ( b - sqrt( Y*Y - a*a) ) /2;
    K = ( a - sqrt( Y*Y - b*b) ) /2;
    DMax = sqrt(L * L + K * K);
    if( d >= DMax ) return false;
    return true;
}
```

Closest Pair Problem:

```
// p contains the point, s1, and s2 are auxiliary arrays
point p[MM], s1[MM], s2[MM];
bool sortX(point &a, point &b) { return ( a.x == b.x ) ? a.y < b.y : a.x < b.x; }
bool sortY(point &a, point &b) { return ( a.y == b.y ) ? a.x < b.x : a.y < b.y; }
double closestPair( int k1, int k2 ){
    double d, d2 ,d3;
    if(k2-k1+1 == 1) return 0;
    if(k2-k1+1 == 2) return Distance(p[k1], p[k2]);
    if(k2-k1+1 == 3) {
        d  = Distance( p[k1], p[k1+1] );
        d2 = Distance( p[k1+1], p[k1+2]);
        d3 = Distance( p[k1+2], p[k1]);
        return min( min(d, d2), d3 );
    }
    int k, i, j, ns1, ns2;
    k = (k1 + k2) / 2;
    d  = closestPair(k1 , k);
    d2 = closestPair(k+1 , k2);
    if(d > d2)  d = d2;
    ns1 = ns2 = 0;
    for(i = k; i>=k1 ; i--) {
        if( p[k].x - p[i].x  >  d ) break;
        s1[ ns1++ ] = p[i];
    }
    sort(s1, s1+ns1, sortY);
    for(i = k+1; i<=k2 ; i++) {
        if( p[i].x - p[k].x  >  d ) break;
        s2[ ns2++ ] = p[i];
    }
    sort(s2, s2+ns2, sortY);
    for(i=0;i<ns1;i++) {
        for(j=0;j<ns2;j++) {
            if(s2[j].y - s1[i].y > d) break;
            d = min( d, Distance( s1[i], s2[j] ) );
        }
    }
    return d;
}
int main() {
    //take n, points in p
    sort( p, p+n, sortX );
    double d = closestPair(0,n-1);
    return 0;
}
```

Misc Geometric Formula:

Triangle	Circum Radius = a*b*c/(4*area) In Radius = area/s, where s = (a+b+c)/2 length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2 length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)
Ellipse	Area = PI*a*b Circumference = 4a *int(0,PI/2){sqrt(1-(k*sint)*(k*sint))}dt = 2*PI*sqrt((a*a+b*b)/2) approx where k = sqrt((a*a-b*b)/a) = PI*(3*(r1+r2)-sqrt[(r1+3*r2)*(3*r1+r2)])
Spherical cap	V = (1/3)*PI*h*h*(3*r-h) Surface Area = 2*PI*r*h
Spherical Sector	V = (2/3)*PI*r*r*h
Spherical Segment	V = (1/6)*PI*h*(3*a*a+3*b*b+h*h)
Torus	V = 2*PI*PI*R*r*r
Truncated Conic	V = (1/3)*PI*h*(a*a+a*b+b*b) Surface Area = PI*(a+b)*sqrt(h*h+(b-a)*(b-a)) = PI*(a+b)*l
Pyramidal frustum	(1/3)*h*(A1+A2+sqrt(A1*A2))

Misc Trigonometric Functions and Formulas:

tan A/2 = ±sqrt((1-cos A)/(1+cos A))
= sin A / (1+cos A)
= (1-cos A) / sin A
= cosec A - cot A

sin 3A = 3*sin A - 4*sincube A cos 3A = 4*coscube A - 3*cos A
tan 3A = (3*tan A-tancube A)/(1-3*tansq A)
sin 4A = 4*sin A*cos A - 8*sincube A*cos A
cos 4A = $8*\cos^4 A - 8*\text{cossq } A + 1$
$[r*(\text{cost}+i*\text{sint})]^p = r^p*(\cos pt+i*\sin pt)$
acos**x** + **b**sin**x** = **c**, x = 2nπ + α ± β, where
cosα = a / (sqrt(a^2+b^2)), cosβ = c / (sqrt(a^2+b^2));

2sinAcosB = sin(A+B) + sin(A-B)
2cosAsinB = sin(A+B) - sin(A-B)
2cosAcosB = cos(A-B) + cos(A+B)
2sinAsinB = cos(A-B) - cos(A+B)
sinC + sinD = 2sin[(C+D)/2]cos[(C-D)/2]
sinC - sinD = 2cos[(C+D)/2]sin[(C-D)/2]
cosD + cosC = 2cos[(C+D)/2]cos[(C-D)/2]
cosD - cosC = 2sin[(C+D)/2]sin[(C-D)/2]

Misc Integration Formula:

a^x => a^x/ln(a)
1/sqrt(x*x+a*a) => ln(x+sqrt(x*x+a*a))
1/sqrt(x*x-a*a) => ln(x+sqrt(x*x-a*a))
1/(x*sqrt(x*x+a*a) => -(1/a)*ln([a+sqrt(x*x+a*a)]/x)
1/(x*sqrt(a*a-x*x) => -(1/a)*ln([a+sqrt(a*a-x*x)]/x)

Misc Differentiation Formula:

asin x => 1/sqrt(1-x*x) acos x => -1/sqrt(1-x*x)
atan x => 1/(1+x*x) acot x => -1/(1+x*x)
asec x => 1/[x*sqrt(x*x-1)] acosec x => -1/[x*sqrt(x*x-1)]
a^x => a^x*ln(x) cot x => -cosecsq x
sec x => sec x * tan x cosec x => -cosec x * cot x

Centroid of a 2D polygon:

As in the calculation of the area above, xN is assumed to be x0, in other words the polygon is closed.

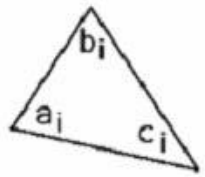
$$c_x = \frac{1}{6A}\sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$c_y = \frac{1}{6A}\sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Centroid of a 3D shell described by 3 vertex facets:

The centroid C of a 3D object made up of a collection of N triangular faces with vertices (a_i,b_i,c_i) is given below. R_i is the average of the vertices of the i'th face and Ai is twice the area of the i'th face. Note the faces are assumed to be thin sheets of uniform mass, they need not be connected or form a solid object.

$$C = \frac{\sum_{i=0}^{N-1} A_i R_i}{\sum_{i=0}^{N-1} A_i}$$



$$R_i = (a_i + b_i + c_i) / 3$$

$$A_i = \|(b_i - a_i) \otimes (c_i - a_i)\|$$

Mirror point(mx,my) of a point(x,y) w.r.to a line(ax+by+c=0):

```
void mirrorPoint(double a,double b,double c,double x,double y,double &mx,double &my) {
    mx = - x*(a*a-b*b) - 2.0*a*b*y - 2.0*a*c;   mx /= (a*a+b*b);
    my = y*(a*a-b*b) - 2.0*a*b*x - 2.0*b*c; my /= (a*a+b*b);
}
```

Circum Circle:

```
R = abc / (4*area)
//measuring the Circum_center M(x,y):
k1 = A.x*A.x - B.x*B.x + A.y*A.y - B.y*B.y;
k2 = A.x*A.x - C.x*C.x + A.y*A.y - C.y*C.y;
k3 = (A.x*C.y + B.x*A.y + C.x*B.y) - (C.x*A.y + A.x*B.y + B.x*C.y);
M.x = (k2*(A.y-B.y) - k1*(A.y-C.y))/(2.*k3);
M.y = (k1*(A.x-C.x) - k2*(A.x-B.x))/(2.*k3);
```

In Circle:

```
// The triangle consists of points A, B and C
r   = area / s
I.x = (A.x*a + B.x*b + C.x * c) / (a+b+c)
I.y = (A.y*a + B.y*b + C.y * c) / (a+b+c)
```

Great circle Distance Between 2 points given in Longitude/latitude format [Radius = R]

```
haversine(x) = ( 1 - cos(x) )/2.0;
a = haversine(lat2 - lat1)
b = cos(lat1) * cos(lat2) * haversine(lon2 - lon1)
c = 2 * atan2(sqrt(a + b), sqrt(1 - a - b))
d = R * c
```

Determining if a point lies on the interior of a 3D convex polygon:

```
// To determine whether a point is on the interior of a convex polygon in 3D, one
// might be tempted to first determine whether the point is on the plane, then
// determine its interior status. Both of these can be accomplished at once by
// computing the sum of the angles between the test point (q below) and every pair of
// edge points p[i]->p[i+1]. This sum will only be twopi if both the point is on the
// plane of the polygon AND on the interior. The angle sum will tend to 0 the further
// away from the polygon point q becomes. The following code snippet returns the angle
// sum between the test point q and all the vertex pairs. The angle sum is in radians.
#define EPSILON 0.0000001
#define MODULUS(p) (sqrt(p.x*p.x + p.y*p.y + p.z*p.z))
const double TWOPI = 6.2831853071795864769252867, RTOD = 57.2957795;
double CalcAngleSum( point3D q, point3D *p, int n ) {
    double m1,m2,anglesum=0,costheta;
    point3D p1, p2;
    for(int i=0;i<n;i++){
        p1.x = p[i].x - q.x; p1.y = p[i].y - q.y; p1.z = p[i].z - q.z;
        p2.x = p[(i+1)%n].x - q.x;
        p2.y = p[(i+1)%n].y - q.y;
        p2.z = p[(i+1)%n].z - q.z;
        m1 = MODULUS(p1), m2 = MODULUS(p2);
        if(m1*m2 <= EPSILON) return(TWOPI); // We are on a node, consider this inside
        else costheta = (p1.x*p2.x + p1.y*p2.y + p1.z*p2.z) / (m1*m2);
        anglesum += acos(costheta);
    }
    return(anglesum);
}
```

Rotation Matrices:

$$Q_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, Q_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, Q_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, Q_{2\times2} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

Construct n from the Sum of Its Divisors:

```
// powi64(a, b) computes a^b, rememver that prime upto i-1 are used
i64 table[NN+1][NN+1]; // if there is an overflow, table[i][j] = inf;
void preprocessTable() {
    for( int i = 0; i <= NN; i++ ) table[0][i] = 1;
    for( int i = 1; i <= NN; i++ ) {
        table[i][0] = 1;
        for( int j = 1; j < NN; j++ ) table[i][j] = table[i][j-1] + powi64(pr[i-1], j);
    }
}
vector <i64> calculateXFromSumOfDivisors( int sum ) {
    vector <i64> res;
    i64 val = 1, prevD = 1;
    for( int i = NN; ; i-- ) {
        if( sum == 1 ) {
            res.push_back( val ); // Here the value is saved
            sum *= prevD, val = 1;
        }
        if( i <= 0 || sum == 1 ) break;
        for( int j = NN - 1; j >= 0; j-- ) {
            if( table[i][j] > 1 && ( sum % table[i][j] == 0 ) ) {
                val *= powi64( pr[i-1], j );
                sum /= table[i][j], prevD = table[i][j];
                break;
            }
        }
    }
    return res;
}
```

Misc Geometry:

```
const double eps = 1e-11, pi = 2 * acos( 0.0 );
struct point { // Creates normal 2D point
    double x, y;
    point() {}
    point( double xx, double yy ) { x = xx, y = yy; }
};
struct point3D { // Creates normal 3D point
    double x, y, z;
};
struct line { // Creates a line with equation ax + by + c = 0
    double a, b, c;
    line() {}
    line( point p1,point p2 ) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
    }
};
struct circle { // Creates a circle with point 'center' as center and r as radius
    point center;
    double r;
    circle() {}
    circle( point P, double rr ) { center = P; r = rr; }
};
struct segment { // Creates a segment with two end points -> A, B
    point A, B;
    segment() {}
    segment( point P1, point P2 ) { A = P1, B = P2; }
};
```

```
inline bool eq(double a, double b) { return fabs( a - b ) < eps; } //two numbers are equal
```

Distance - Point, Point:

```
inline double Distance( point a, point b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) );
}
```

Distance^2 - Point, Point:

```
inline double sq_Distance( point a, point b ) {
    return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
}
```

Distance - Point, Line:

```
inline double Distance( point P, line L ) {
    return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b );
}
```

Is left Function:

```
inline double isleft( point p0, point p1, point p2 ) {
    return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y - p0.y ) );
}
```

Intersection - Line, Line:

```
inline bool intersection( line L1, line L2, point &p ) {
    double det = L1.a * L2.b - L1.b * L2.a;
    if( eq ( det, 0 ) ) return false;
    p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
    p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
    return true;
}
```

Intersection - Segment, Segment:

```
inline bool intersection( segment L1, segment L2, point &p ) {
    if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
        return false; // can lie on another, just check their equations, and check overlap
    }
    return(eq(Distance(L1.A,p)+Distance(L1.B,p),Distance(L1.A,L1.B)) &&
        eq(Distance(L2.A,p)+Distance(L2.B,p),Distance(L2.A,L2.B)));
}
```

Perpendicular Line of a Given Line Through a Point:

```
inline line findPerpendicularLine( line L, point P ) {
    line res; //line perpendicular to L, and intersects with P
    res.a = L.b, res.b = -L.a;
    res.c = -res.a * P.x - res.b * P.y;
    return res;
}
```

Distance - Point, Segment:

```
inline double Distance( point P, segment S ) {
    line L1 = line(S.A,S.B), L2; point P1;
    L2 = findPerpendicularLine( L1, P );
    if( intersection( L1, L2, P1 ) )
        if( eq ( Distance( S.A, P1 ) + Distance( S.B, P1 ), Distance( S.A, S.B ) ) )
            return Distance(P,L1);
    return min ( Distance( S.A, P), Distance( S.B, P) );
}
```

Area of a 2D Polygon:

```
double areaPolygon( point P[], int n ) {
    double area = 0;
    for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y - P[j].y * P[i].x;
    return fabs(area)/2;
}
```

Point Inside Polygon:

```
bool insidePoly( point &p, point P[], int n ) {
    bool inside = false;
    for( int i = 0, j = n - 1; i < n; j = i++ )
        if( (( P[i].x < p.x ) ^ ( P[j].x < p.x )) &&
            (P[i].y - P[j].y) * abs(p.x - P[j].x) < (p.y - P[j].y) * abs(P[i].x - P[j].x) )
            inside = !inside;
    return inside;
}
```

Intersection - Circle, Line:

```
inline bool intersection(circle C,line L,point &p1,point &p2) {
    if( Distance( C.center, L ) > C.r + eps ) return false;
    double a, b, c, d, x = C.center.x, y = C.center.y;
    d = C.r*C.r - x*x - y*y;
    if( eq( L.a, 0) ) {
        p1.y = p2.y = -L.c / L.b;
        a = 1;
        b = 2 * x;
        c = p1.y * p1.y - 2 * p1.y * y - d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs (d) );
        p1.x = ( b + d ) / ( 2 * a );
        p2.x = ( b - d ) / ( 2 * a );
    }
    else {
        a = L.a *L.a + L.b * L.b;
        b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
        c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs(d) );
        p1.y = ( b + d ) / ( 2 * a );
        p2.y = ( b - d ) / ( 2 * a );
        p1.x = ( -L.b * p1.y -L.c ) / L.a;
        p2.x = ( -L.b * p2.y -L.c ) / L.a;
    }
    return true;
}
```

Find Points that are r1 unit away from A, and r2 unit away from B:

```
inline bool findpointAr1Br2(point A,double r1,point B, double r2,point &p1,point &p2) {
    line L;
    circle C;
    L.a = 2 * (B.x - A.x );
    L.b = 2 * (B.y - A.y );
    L.c = A.x * A.x + A.y * A.y - B.x * B.x - B.y * B.y + r2 * r2 - r1 * r1;
    C.center = A;
    C.r = r1;
    return intersection( C, L, p1, p2 );
}
```

Intersection Area between Two Circles:

```
inline double intersectionArea2C( circle C1, circle C2 ) {
    C2.center.x = Distance( C1.center, C2.center );
    C1.center.x = C1.center.y = C2.center.y = 0;
    if( C1.r < C2.center.x - C2.r + eps ) return 0;
    if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
    if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
    double c, CAD, CBD, res;
    c = C2.center.x;
    CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
    CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
    res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin ( CBD ) );
    return .5 * res;
}
```

Circle Through Thee Points:

```
circle CircleThrough3points( point A, point B, point C) {
    double den; circle c;
    den = 2.0 *((B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x));
    c.center.x =( (C.y-A.y)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y) -
                (B.y-A.y)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) );
    c.center.x /= den;
    c.center.y =( (B.x-A.x)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) -
                (C.x-A.x)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y) );
    c.center.y /= den;
    c.r = Distance( c.center, A );
    return c;
}
```

Rotating a Point anticlockwise by 'theta' radian w.r.t Origin:

```
inline point rotate2D( double theta, point P ) {
    point Q;
    Q.x = P.x * cos( theta ) - P.y * sin( theta );
    Q.y = P.x * sin( theta ) + P.y * cos( theta );
    return Q;
}
```

Convex Hull (Graham Scan) O(nlogn):

```
// compare Function for qsort in convex hull
point Firstpoint;
int cmp(const void *a,const void *b) {
    double x,y;
    point aa,bb;
    aa = *(point *)a;
    bb = *(point *)b;
    x = isleft( Firstpoint, aa, bb );
    if( x > eps ) return -1;
    else if( x < -eps ) return 1;
    x = sq_Distance( Firstpoint, aa );
    y = sq_Distance( Firstpoint, bb );
    if( x + eps < y ) return -1;
    return 1;
}
// 'P' contains all the points, 'C' contains the convex hull
// 'nP' = total points of 'P', 'nC' = total points of 'C'
void ConvexHull( point P[], point C[], int &nP, int &nC ) {
    int i, j, pos = 0; // Remove duplicate points if necesary
    for( i = 1; i < nP; i++ )
        if( P[i].y < P[pos].y || ( eq( P[i].y, P[pos].y ) && P[i].x > P[pos].x + eps ) )
            pos = i;
    swap( P[pos], P[0] );
    Firstpoint = P[0];
    qsort( P + 1, nP - 1, sizeof( point ), cmp );
    C[0] = P[0]; C[1] = P[1];
    i = 2, j = 1;
    while( i < nP ) {
        if( isleft( C[j-1], C[j], P[i] ) > -eps ) C[++j] = P[i++];
        else j--;
    }
    nC = j + 1;
}
```

Angle between Vectors:

```
inline double angleBetweenVectors( point O, point A, point B ) { // vector OA to OB
    point t1, t2;
    t1.x = A.x - O.x; t1.y = A.y - O.y;
    t2.x = B.x - O.x; t2.y = B.y - O.y;
    double theta = (atan2(t2.y, t2.x) - atan2(t1.y, t1.x));
    if( theta < 0 ) theta += 2 * pi;
    return theta;
}
```

Printing a Polynomial:

```
void printPoly( int coeffs[], int n, const char *var ) {
    bool empty = true;
    while( n-- ) {
        if( coeffs[n] || empty && !n ) {
            if(!empty || coeffs[n]<0 ) printf("%c",(coeffs[n]>0?'+' : '-' ) );
            if( abs( coeffs[n] ) > 1 || n == 0 ) printf( "%d", abs( coeffs[n] ) );
            if( n > 0 ) {
                if( abs( coeffs[n] ) > 1 ) printf( "*" );
                printf( "%s", var );
            }
            if( n > 1 ) printf( "^%d", n );
            empty = false;
        }
    }
}
```

Finding Determinant:

```
/* We have found the Minimum col in 1st row.
Then subTract from All nonZero column the minimum Column as possible.
Such as: 5 8 7 - 3 2 Then in next Step in stead of 5 we start with 3 because Modulus
must less than divider(5). */
#define MAX 50
#define INF 32000
int mat[MAX][MAX], N, mul;
void xchgColumn( int i, int j ) {
    for( int k = 0; k < N; k++ ) swap( mat[k][i], mat[k][j] );
    mul *= -1;
}
void reduceMat(void){
    int i, j, minCol, min, absMin, nonZero = 0, absValue, d, r;
    for(absMin=INF,i=0;i<N;i++){
        if(mat[0][i]){
            nonZero++;
            if(mat[0][i] < 0) absValue = -mat[0][i];
            else absValue = mat[0][i];
            if(absValue < absMin){
                absMin = absValue;
                min    = mat[0][i];
                minCol = i;
            }
        }
    }
    if(!nonZero) { mul = 0; return; }
    while(nonZero > 1){
        for(i=0;i<N;i++){
            if(i != minCol && mat[0][i]){
                d = mat[0][i]/min;  r = mat[0][i]-d*min;
                for(j=0;j<N;j++) mat[j][i]=mat[j][i]-d*mat[j][minCol];
                if(r){ min = r; minCol = i; }
                else nonZero--;
            }
        }
    }
    for(i=0;!mat[0][i];i++);
    if(i!=0) xchgColumn(0,i);
    mul *= mat[0][0];
    for(i=1;i<N;i++) for(j=1;j<N;j++) mat[i-1][j-1] = mat[i][j];
    N--;
}
int main() {
    int i,j,result;
    while(scanf("%d",&N) && N){
        for(i=0;i<N;i++)
            for(j=0;j<N;j++)
                scanf("%d",&mat[i][j]);
        if(N > 1){
            mul = 1;
            while(N > 2 && mul) reduceMat();
            result = mat[0][0]*mat[1][1]-mat[0][1]*mat[1][0];
            result = result*mul;
            printf("%d\n",result);
        }
        else printf("%d\n",mat[0][0]);
    }
    return 0;
}
```

Aho Corasick:

```
const int MM = 100005, NN = 1005;   // MM - long string length, NN - small string length
const int MAXCHAR = 52, MAX = 200000;   // MAXCHAR maximum characters, MAX maximum nodes
char T[MM], a[NN][NN];                  // Long string T, small strings a[]
int val( char ch ) { if( ch >= 'a' ) return ch - 97; else return ch - 39; }
struct Trie {
    int N;              // Contains the number of nodes of the Trie
    struct node {
        int edge[MAXCHAR], f;   // The alphabets, f failure function
        bool out; // out function, gives the set of patterns recognized entering this state
        void clear() {          // Clears the node
            memset( edge, -1, sizeof( edge ) );
            f = out = false;
        }
    }P[MAX];
    void clear() {      // Clears the Trie, Initially g( 0, x ) = 0, for all x
        N = 1; P[0].clear();
        memset( P[0].edge, 0, sizeof( P[0].edge ) );
    }
    void insert( char *a ) {            // Inserts an element into the trie
        int p = 0, i, k;
        for( i = 0; a[i]; i++ ) {
            k = val( a[i] );
            if( P[p].edge[k] <= 0 ) {   // Edge is not available
                P[p].edge[k] = N;
                P[N++].clear();         // Clear the edge, and increase N
            }
            p = P[p].edge[k];           // Go to the next edge
        }
        P[p].out = true;
    }
    void computeFailure() {             // Computes the failure function
        int i, u, v, w;
        queue <int> Q;
        for( i = 0; i < MAXCHAR; i++ ) if( P[0].edge[i] ) {
            u = P[0].edge[i]; P[u].f = 0; Q.push(u);
        }
        while( !Q.empty() ) {
            u = Q.front(); Q.pop();
            for( i = 0; i < MAXCHAR; i++ ) if( P[u].edge[i] != -1 ) {
                v = P[u].edge[i];
                Q.push(v);
                w = P[u].f;
                while( P[w].edge[i] == -1 ) w = P[w].f;
                w = P[v].f = P[w].edge[i];
                P[v].out |= P[w].out;
            }
        }
    }
}A;
int n, cases;
bool mark[MAX];
void AhoCorasick( Trie &A, char *T ) {// Finds the occurences of strings in the trie, in T
    int i, q = 0, k; // q Initial State
    for( i = 0; i < A.N; i++ ) mark[i] = false;
    for( i = 0; T[i]; i++ ) {
        k = val( T[i] );
        while( A.P[q].edge[k] == -1 ) q = A.P[q].f;
        q = A.P[q].edge[k];
        int x = q;
        if( A.P[x].out && !mark[x] ) {
            mark[x] = true;
            x = A.P[x].f;
        }
    }
}
```

```
bool exists( Trie &A, char *a ) {
    int i, q = 0, k;
    for( i = 0; a[i]; i++ ) {
        k = val(a[i]);
        q = A.P[q].edge[k];
    }
    return mark[q];
}
int main() {
    scanf("%s %d", T, &n);
    A.clear();
    for( int i = 0; i < n; i++ ) {
        scanf("%s", a[i]);
        A.insert( a[i] );
    }
    A.computeFailure();
    AhoCorasick( A, T );
    for( int i = 0; i < n; i++ ) {
        if( exists( A, a[i] ) ) puts("y");
        else puts("n");
    }
    return 0;
}
```

Area of a 3D Polygon:

```
double area3D_Polygon( int n, point3D *V ) {
    double area = 0;
    double an, ax, ay, az; // abs value of normal and its coords
    int coord; // coord to ignore: 1=x, 2=y, 3=z
    int i, j, k;
    double val;
    point3D u, v, N;
    // Finding Unit Normal Vector
    u.x = V[1].x - V[0].x; u.y = V[1].y - V[0].y; u.z = V[1].z - V[0].z;
    v.x = V[2].x - V[0].x; v.y = V[2].y - V[0].y; v.z = V[2].z - V[0].z;
    N.x = u.y * v.z- u.z * v.y;
    N.y = u.z * v.x- u.x * v.z;
    N.z = u.x * v.y- u.y * v.x;
    val = sqrt( N.x * N.x + N.y * N.y + N.z * N.z );
    N.x /= val; N.y /= val; N.z /= val;
    // select largest abs coordinate to ignore for projection
    ax = (N.x > 0 ? N.x : -N.x);     // abs x-coord
    ay = (N.y > 0 ? N.y : -N.y);     // abs y-coord
    az = (N.z > 0 ? N.z : -N.z);     // abs z-coord
    coord = 3;                       // ignore z-coord
    if(ax > ay) {
        if (ax > az) coord = 1;      // ignore x-coord
    }
    else if(ay > az) coord = 2;      // ignore y-coord
    // compute area of the 2D projection
    for( i = 1, j = 2, k = 0; i <= n; i++, j++, k++) {
        switch (coord) {
            case 1: area += (V[i].y * (V[j].z - V[k].z)); continue;
            case 2: area += (V[i].x * (V[j].z - V[k].z)); continue;
            case 3: area += (V[i].x * (V[j].y - V[k].y)); continue;
        }
    }
    // scale to get area before projection
    an = sqrt( ax*ax + ay*ay + az*az);  // length of normal vector
    switch (coord) {
        case 1: area *= (an / (2*ax)); break;
        case 2: area *= (an / (2*ay)); break;
        case 3: area *= (an / (2*az));
    }
    return fabs( area );
}
```

To Roman:

```
string toRoman( int n ) {
    if( n < 4 ) return fill( 'i', n );
    if( n < 6 ) return fill( 'i', 5 - n ) + "v";
    if( n < 9 ) return string( "v" ) + fill( 'i', n - 5 );
    if( n < 11 ) return fill( 'i', 10 - n ) + "x";
    if( n < 40 ) return fill( 'x', n / 10 ) + toRoman( n % 10 );
    if( n < 60 ) return fill( 'x', 5 - n / 10 ) + 'l' + toRoman( n % 10 );
    if( n < 90 ) return string( "l" )+fill('x',n/10-5) + toRoman( n % 10 );
    if( n < 110 ) return fill( 'x', 10 - n / 10 ) + "c" + toRoman( n % 10 );
    if( n < 400 ) return fill( 'c', n / 100 ) + toRoman( n % 100 );
    if( n < 600 ) return fill( 'c', 5 - n / 100 ) + 'd' + toRoman( n % 100 );
    if( n < 900 ) return string("d")+fill('c',n/100- 5 ) + toRoman( n % 100 );
    if( n < 1100 ) return fill( 'c', 10 - n / 100 ) + "m" + toRoman( n % 100 );
    if( n < 4000 ) return fill( 'm', n / 1000 ) + toRoman( n % 1000 );
    return "?";
}
```

Binary Indexed Tree:

```
void insert(int x, int v) {
    while( x <= n ) {
        bit[x] += v;
        x += x & -x;
    }
}
int readRes( int x ) {
    int ret = 0;
    while( x > 0 ){
        ret += bit[x];
        x -= x & -x;
    }
    return ret;
}
```

Shank's Algorithm:

This algorithm finds x (0 <= x <= p - 2) for the equation

$b = a^x \bmod p$ where b, a, p are known

Using the fact that x can be expressed as jm + i, where 0 <= i <= m - 1, 0 <= j < p/m, and m = ceil(sqrt(p - 1))

So, the equation can be written as

$b = a^{mj+i} \bmod p$

$b = a^{mj}\ a^{i} \bmod p$

$ba^{-i} = a^{mj} \bmod p$

If two lists of ordered pairs (i, ba^{-i}) and (j, a^{mj}), ordered by their second components are built, then it is possible to find one pair from each list that have equal second components. Then x = mj + i, where i and j are the first elements of the matching pairs.

Negative Base:

```
string negaBase(int n,int b){
    int i,tmp;
    string a;
    for(i=0;n;i++){
        tmp=n%b; n=n/b;
        if(tmp<0) {     tmp+= (-b), n++;        }
        a+='0'+tmp;
    }
    for(n=0;n<(i/2);n++) swap(a[n],a[i -n - 1]);
    if(i) return a;
    return "0";
}
```

Joseph:

```
int joseph(int n,int k){
    if(n==1) return 0;
    return ((joseph(n-1,k)+k)%n);
}
```

Problem name : Mission Impossible

```
// Given some circles(integer co-ordinate,integer radius) and points in (0,0)->(1000,1000)
// Finds the way to the point from infinity without touching any circle or entering it

struct node{   double x, y;   }border[1010];
struct circle{   int x, y, r;     }cir[ 35 ];
double cost[ 1010 ];
char grid[ 2010 ][ 2010 ];
short spypos[ 2010 ][ 2010 ];
int mmx;
struct pts{    short x, y;   }tt;
bool spyflag[ 1010 ];

stack <pts> stk;
int dx[ ] = { -1,-1,-1,1,1,1,0,0};
int dy[ ] = { 0,1,-1,0,1,-1,1,-1};
bool vis[ 2010 ][ 2010 ];

double dis( node a, node b ){ return sqrt( ( a.x - b.x ) *( a.x - b.x ) + ( a.y - b.y ) * (
a.y - b.y ) ) + 1e-9;}
double cdis( double x, double y, circle b ){ return sqrt( ( x - b.x ) *( x - b.x ) + ( y -
b.y ) * ( y - b.y ) ) + 1e-9;}
double dist(node c, node a, node b) // line segment -> point distance| c is the point
{
    double distanceSegment, distanceLine;
    double r_numerator = (c.x-a.x)*(b.x-a.x) + (c.y-a.y)*(b.y-a.y);
    double r_denomenator = (b.x-a.x)*(b.x-a.x) + (b.y-a.y)*(b.y-a.y);
    double r = r_numerator / r_denomenator;

    double px = a.x + r*(b.x-a.x);
    double py = a.y + r*(b.y-a.y);

    double s =  ((a.y-c.y)*(b.x-a.x)-(a.x-c.x)*(b.y-a.y) ) / r_denomenator;
    distanceLine = fabs(s)*sqrt(r_denomenator);
    double xx = px;
    double yy = py;
    if ( (r >= 0) && (r <= 1) ) distanceSegment = distanceLine;
    else{
        double dist1 = (c.x-a.x)*(c.x-a.x) + (c.y-a.y)*(c.y-a.y);
        double dist2 = (c.x-b.x)*(c.x-b.x) + (c.y-b.y)*(c.y-b.y);
        if (dist1 < dist2){
    xx = a.x;
    yy = a.y;
    distanceSegment = sqrt(dist1);
        }
        else{
    xx = b.x;
    yy = b.y;
    distanceSegment = sqrt(dist2);
        }
    }
    return distanceSegment;
}

bool valid( int x, int y ){
    if( x < 0 || x > mmx || y < 0 || y > mmx ) return 0;
    if( vis[ x ][ y ] ) return 0;
4   if( grid[ x ][ y ] == 1 ) return 0;
    return 1;
}
void bfs(){
    int i, mx =0;
    while( !stk.empty() ){
        pts now = stk.top();
        stk.pop();
```

```
        spyflag[ spypos[ now.x ][ now.y ] ] = 1;
        for(i = 0; i < 8 ; i ++ )
        if( valid( now.x + dx[ i ], now.y + dy[ i ] ) ){
            tt.x = now.x + dx[ i ];
            tt.y = now.y + dy[ i ];
            vis[ tt.x ][ tt.y ] = 1;
            stk.push( tt );
        }
    }
}

int main(){
    int B, N, M, i, j, k;
    while( cin >> B ){
        if( !B ){
            cin >> B;
            cin >> B;
            break;
        }
        CLR( grid );
        CLR( spypos );
        CLR( spyflag );
        CLR( vis );
        while( !stk.empty() ) stk.pop();
        for(i = 0 ; i < B ; i ++ ) scanf("%lf %lf", &border[ i ].x,  &border[ i ].y );
        border[ B ] = border[ 0 ];

        cin >> N;
        int x, y;
        for(i = 0 ; i < N ; i ++ ){
            scanf("%d %d", &x,  &y );
            cost[ i ] = 1e15;
            for(j = 0; j < B  ; j ++ ){
                node nn;
                nn.x = x, nn.y = y;
                double now = dist( nn, border[ j ], border[ j + 1 ]);
                if( now < cost[ i ] )   cost[ i ] = now;
            }
            spypos[ x * 2 ][ y * 2] = i + 1;
            if( x * 2 > mmx ) mmx = x * 2 ;
            if( y * 2 > mmx ) mmx = y * 2 ;
        }
        cin >> M;
        for(i = 0 ; i < M ; i ++ ) {
            scanf("%d %d %d", &cir[ i ].x,  &cir[ i ].y , &cir[ i ].r);
            cir[ i ].x *= 2,  cir[ i ].y *= 2 , cir[ i ].r *= 2;
            if( cir[ i ].x > mmx ) mmx = cir[ i ].x;
            if( cir[ i ].y > mmx ) mmx = cir[ i ].y;
        }
        mmx += 10;
        if( mmx > 2000 ) mmx = 2000;

        for(k = 0; k < M ; k ++ ){
            int stx = cir[ k ].x;
            if( stx < 0 ) stx = 0;

            int sty = cir[ k ].y;
            if( sty < 0 ) sty = 0;
            double rr = (double)cir[ k ].r;
            for(i = stx; i <= mmx; i++ ){
                for(j = sty ; j <= mmx;j++)
                    if(cdis(i, j, cir[ k ] ) <= rr ) grid[ i ][ j ] = 1;
                    else break;
                for(j = sty ; j >=0 ; j--)
                    if(cdis(i, j, cir[ k ] ) <= rr ) grid[ i ][ j ] = 1;
                    else break;
```

```
            }
            for(i = stx; i >= 0 ; i -- ){
                for(j = sty ; j <= mmx;j++)
                    if(cdis(i, j, cir[ k ] ) <= rr ) grid[ i ][ j ] = 1;
                    else break;
                for(j = sty ; j >= 0 ; j--)
                    if(cdis(i, j, cir[ k ] ) <= rr ) grid[ i ][ j ] = 1;
                    else break;
            }
        }
        for(j = 0; j <= mmx ; j ++ ){
            pts tmp;
            tmp.x =  0, tmp.y = j;
            if( grid[ 0 ][ j ] == 0 ){
                stk.push( tmp );
                vis[ 0 ][ j ] = 1;
            }
            tmp.x =  mmx, tmp.y = j;
            if( grid[ mmx ][ j ] == 0 ){
                stk.push( tmp );
                vis[ mmx ][ j ] = 1;
            }

            tmp.x =  j, tmp.y = 0;
            if( grid[ j ][ 0 ] == 0 ){
                stk.push( tmp );
                vis[ j ][ 0 ] = 1;
            }
            tmp.x =  i, tmp.y = mmx;
            if( grid[ j ][ mmx ] == 0 ){
                stk.push( tmp );
                vis[ j ][ mmx ] = 1;
            }
        }
        bfs();
        int rem = -1;
        double mmm = 0;
        for(i = 1; i <= N; i ++ ){
            if( spyflag[ i ] ){
                if( cost[ i - 1 ] > mmm ){
                    mmm = cost[ i - 1 ];
                    rem = i;
                }
            }
        }
        if( rem == -1 ) printf("Mission impossible\n");
        else printf("Contact informer %d\n",rem);

    }
    return 0;
}
```

Point inside a polygon (convex) O(logn)

```
#define i64 long long
struct point{   i64 x,y;  }conv[100001];
i64 N;
i64 area(i64 x1, i64 y1, i64 x2, i64 y2, i64 a, i64 b){
    i64 k = (x1 - x2)*(y2 - b) - (y1 - y2)*(x2 - a);
    if(k<0) return -k;
    return k;
}
bool inside(i64 x1, i64 y1, i64 x2, i64 y2, i64 x3,i64 y3,i64 a, i64 b){
    i64 aa = area(x1,y1,x2,y2,x3,y3);
    i64 a1 = area(x1,y1,x2,y2,a,b);
    i64 a2 = area(x1,y1,a,b,x3,y3);
    i64 a3 = area(a,b,x2,y2,x3,y3);

    if(aa - a1 - a2 - a3 == 0)  return true;
    return false;
}
bool isleft(i64 x1, i64 y1, i64 x2, i64 y2,i64 a,i64 b){
    i64 k = (x1 - x2)*(y2 - b) - (y1 - y2)*(x2 - a);
    if(k<0) return 0;
    return 1;
}
bool cal(i64 a,i64 b,i64 st,i64 en){
    if(en - st < 2 )    return false;
    if(en - st == 2){
    if(inside(conv[st].x,conv[st].y,conv[st+1].x, conv[st+1].y,conv[en].x,conv[en].y,a,b ))
            return true;
    }
    else if(en - st == 3){
 if(inside(conv[st].x,conv[st].y,conv[st+1].x,conv[st+1].y,conv[st+2].x,conv[st+2].y,a,b ))
            return true;

    if(inside(conv[st].x,conv[st].y,conv[en].x, conv[en].y,conv[st+2].x,conv[st+2].y,a,b ))
            return true;
    }
    else{
        bool p1;
        i64 m = (st+en+1)/2;
        if(inside(conv[st].x,conv[st].y,conv[en].x, conv[en].y,conv[m].x,conv[m].y,a,b ))
            return true;
        if(!isleft(conv[st].x,conv[st].y,conv[m].x, conv[m].y,a,b)){
            p1 = cal(a,b,st,m);
            if(p1 == true)
            return true;
        }
        else{
            p1 = cal(a,b,m,en);
            if(p1 == true)
                return true;
        }
    }
    return false;
}
insidepoly(point conv[],int nC, int a, int b ){
    int N = nC;
    bool res = cal(a,b,0,N-1);
    if(res == true) return true;
    else return false;
}
```

Point Insilde a Polygon O(n) (Convex):

```
struct point{ int x, y; };
inline double isleft( point p0, point p1, point p2 ){
    return ( ( p1.x - p0.x) * ( p2.y - p0.y ) - (p2.x - p0.x ) * ( p1.y - p0.y ) );
}
bool insidepoly(point C[],int nC,point a){
    int i, fl;
    if( isleft(C[ 0 ], C[ 1 ], a ) < 0 ) fl = 1;
    else fl = -1;
    for(i = 0; i < nC; i ++ ){
        int gl;
        if( isleft(C[ i ], C[ i + 1 ], a ) < 0 ) gl = 1;
        else gl = -1;
        if( gl != fl ) return false;
    }
    return true;
}
```

Problem Name: Power Generation: KD - Tree (not optimized):

//inserts function inserts 2-dimensional co-ordinates into tree in O(log n) (average case)
//nsearch finds the nearest neighbour of a given co-ordinate in O(log n)

```
struct point{   int x[2],id; };
struct node{
    point p;
    node *lf,*rt;
    node(){
        lf = rt = 0;
    }
};
int best,id;
node *insert(node *nd,point p,int dim){
    if(nd == NULL){
        node *md = new node();
        md -> p = p;
        return md;
    }
    if(p.x[dim]<nd->p.x[dim]){
        nd->lf = insert(nd->lf,p,!dim);
    }
    else nd->rt = insert(nd->rt,p,!dim);
    return nd;
}
void nsearch(node *nd,point p,int dim){
    if(nd==NULL) return ;
    int d=(nd->p.x[0]-p.x[0])*(nd->p.x[0]-p.x[0])+(nd->p.x[1]-p.x[1])*(nd->p.x[1]-p.x[1]);
    if(d<best || (d==best && nd->p.id <id)){
        id = nd->p.id;
        best = d;
    }
    d = (nd->p.x[dim] - p.x[dim])*(nd->p.x[dim] - p.x[dim]);
    if(d>best){
        if(p.x[dim]<nd->p.x[dim]){
            nsearch(nd->lf,p,!dim);
        }
        else nsearch(nd->rt,p,!dim);
    }
    else{
        nsearch(nd->lf,p,!dim);
        nsearch(nd->rt,p,!dim);
    }
}
```

```
int n,C,c[10005],cnt;
vector< vector<int> > adj;

int solve(int x){

    int val = c[x],i;
    for(i = adj[x].size() -1;i>=0;i--){
        val+=solve(adj[x][i]);
    }
    if(val>=C) {
        cnt++;
        return 0;
    }
    return val;
}
int main(){
    int i;
    while(scanf("%d %d",&n,&C)==2){
        if(n==0 && C==0) break;
        adj = vector< vector<int> > (n+5);
        node *root;
        root = NULL;
        point p;
        scanf("%d %d %d",&p.x[0],&p.x[1],&c[0]);
        p.id = 0;
        root = insert(root,p,0);
        for(i = 1;i<n;i++){
            scanf("%d %d %d",&p.x[0],&p.x[1],&c[i]);
            best = inf;
            p.id = i;
            nsearch(root,p,0);
            adj[id].pb(i);
            root = insert(root,p,0);
        }
        cnt = 0;

        solve(0);
        printf("%d\n",cnt);
    }
    return 0;
}
```

Suffix Array with LCP (n log n):

```
const int MAXN = 2005;
const int MAXL = 22;
int n ,stp,mv,suffix[MAXN],tmp[MAXN];
int sum[MAXN],cnt[MAXN],rank[MAXL][MAXN];
char str[MAXN];
int LCP(int u,int v){
    int ret=0,i;
    for(i = stp; i >= 0; i--){
        if(rank[i][u]==rank[i][v]){
            ret += 1<<i;
            u += 1<<i;
            v += 1<<i;
        }
    }
    return ret;
}
bool equal(int u,int v){
    if(!stp)return str[u]==str[v];
    if(rank[stp-1][u]!=rank[stp-1][v]) return false;
    int a = u + mv < n ? rank[stp-1][u+mv] : -1;
    int b = v + mv < n ? rank[stp-1][v+mv] : -1;
    return a == b ;
}
```

```
void update(){
    int i;
    for(i = 0;i < n; i ++) sum[ i ] = 0;

    int rnk = 0;
    for(i = 0;i < n;i++){
        suffix[ i ] = tmp[ i ];
        if( i&&!equal(suffix[i],suffix[i-1])){
            rank[stp][suffix[i]]=++rnk;
            sum[rnk+1]=sum[rnk];
        }
        else rank[stp][suffix[i]]=rnk;
        sum[rnk+1]++;
    }
}
void Sort(){
    int i;
    for(i = 0; i < n; i ++ ) cnt[ i ] = 0;
    memset(tmp,-1,sizeof tmp);
    for(i = 0 ; i < mv; i ++){
        int idx = rank[ stp - 1 ][ n-i-1 ];
        int x = sum[ idx ];
        tmp[ x + cnt[ idx ] ] = n-i-1;
        cnt[ idx ]++;
    }
    for(i = 0;i < n; i ++ ){
        int idx = suffix[ i ] - mv;
        if(idx<0)continue;
        idx = rank[stp-1][idx];
        int x = sum[ idx ];
        tmp[ x + cnt[ idx ] ] = suffix[ i ] - mv;
        cnt[idx]++;
    }
    update();
    return;
}
bool cmp(const int &a,const int &b){
    if(str[a]!=str[b]) return str[a]<str[b];
    return false;
}
int main(){
    scanf("%d", &n);
    scanf ( "%s", str  );
    int i;
    for(i = 0;i < n;i++) tmp[ i ] = i ;

    sort(tmp,tmp+n,cmp);
    stp = 0;
    update();
    ++stp;
    for( mv = 1; mv < n;  mv <<= 1){
        Sort();
        stp++;
    }
    stp--;
    for(i = 0;i<=stp; i++)  rank[ i ][ n ] = -1;
    int res=0;
    for(i = 1; i < n; i ++)
        res=max(res,LCP(suffix[i],suffix[i-1]));
    printf("%d\n",res);
    return 0;
}
```

Problem Name : Bee Breading

```
//Find the distance between two point in a hexagonal grid
struct point{
    int x,y,z;
    void inpoint(int _x,int _y,int _z){
        x = _x;
        y = _y;
        z = _z;
    }
};
point p3d[8];
point givePoint(i64 p){
    int a;
    i64 n;
    point pt;
    for(a = 2;a<=7;a++){
        n =(i64) (  (sqrt((a - 4)*(a - 4) + 12*p) - a + 4 ) /6.0 + 1e-10) ;
        if(3*n*n + (a - 4)*n !=p) continue;
        pt.inpoint(p3d[a].x*n,p3d[a].y*n,p3d[a].z*n);
        return pt;
    }
    return pt;
}
point hexTo3d(i64 s){
    if(s == 1) return p3d[1];
    s--;
    i64 n = (int)ceil((sqrt(9+12*s) - 3) / 6.0 );
    i64 p = 3*n*n + 3*n, q;
    int i;
    for(i = 0;i<5;i++){
        q = p - n;
        if(p>=s && s>=q) break;
        p =q;
    }
    if(i==5) q = 3*n*n + 3*n;
    point pt1 = givePoint(p);
    point pt2 = givePoint(q),ret = pt2;
    if(i == 5) q = 3*(n-1)*(n-1) + 3*(n-1);
    int d = (pt1.x - pt2.x)/n;
    ret.x+=d*(s - q);
    d = (pt1.y - pt2.y)/n;
    ret.y+=d*(s - q);
    d = (pt1.z - pt2.z)/n;
    ret.z+=d*(s - q);
    return ret;
}
int main(){
    p3d[1].inpoint(0,0,0);
    p3d[2].inpoint(-1,0,1);
    p3d[3].inpoint(-1,-1,0);
    p3d[4].inpoint(0,-1,-1);
    p3d[5].inpoint(1,0,-1);
    p3d[6].inpoint(1,1,0);
    p3d[7].inpoint(0,1,1);
    point p,q;
    i64 a,b;
    int t,cs = 1;
    scanf("%d",&t);
    for(cs = 1;cs<=t;cs++){
        scanf("%lld %lld",&a,&b);
        p = hexTo3d(a);
        q = hexTo3d(b);
        printf("Case %d: %d\n",cs,(abs(p.x - q.x) + abs(p.y - q.y) + abs(p.z - q.z)) / 2);
    }
    return 0;
}
```

Minimum Flow:

```
//Given the lower bound and upper bound of the edges, find minimum flow

int cap[105][105],prev[105],flow[105][105],N;
bool bfs(int s,int t){
    int i,u;
    SET(prev);
    prev[s]=-2;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        u=q.front();
        q.pop();
        for(i=0;i<N;i++){
            if(prev[i]==-1 && cap[u][i]>flow[u][i]){
                q.push(i);
                prev[i]=u;
            }
        }
    }
    return prev[t]==-1?false:true;
}
bool dinic(int s,int t){
    int i,u,v,flw;
    CLR(flow);
    while(bfs(s,t)){
        for(i=0;i<N;i++){
            if(prev[i]>=0 && cap[i][t] - flow[i][t]<=0) continue;
            flw=cap[i][t]-flow[i][t];
            for(v=i,u=prev[v];u>=0;v=u,u=prev[v]) flw=min(flw,cap[u][v]-flow[u][v]);
            for(v=i,u=prev[v];u>=0;v=u,u=prev[v]) {
                flow[u][v]+=flw;
                flow[v][u]-=flw;
            }
            flow[i][t]+=flw;
            flow[t][i]-=flw;

        }
    }
    for(i = 0;i<N-2;i++){
        if(cap[s][i]!=flow[s][i]) return false;
    }
    return true;
}
int main(){
    int n,m,i,j,low,x[5000],y[5000],c[5000],f[5000],inf = 1<<29,ans,mid,hi,s[5000];
    while(cin>>n>>m){
        N = n+2;
        for ( i = 0;i<N;i++){
            for(j = 0;j<N;j++){
                cap[i][j] = 0;
            }
        }
        for( i = 0;i<m;i++){
            cin>>x[i]>>y[i]>>c[i]>>f[i];
            x[i] -- ;
            y[i] --;
            cap[x[i]][y[i]] = c[i];
            if(f[i]){
                cap[n][y[i]] += c[i];
                cap[x[i]][n+1] += c[i];
                cap[x[i]][y[i]] -= c[i];

            }
        }
```

```
        cap[n-1][0] = inf;
        if(!dinic(n,n+1)){
            cout<<"Impossible"<<endl;
            continue;
        }
        low = 0; hi = inf;
        while(low<=hi){
            mid = (low+hi) / 2;
            cap[n-1][0] = mid;
            if(dinic(n,n+1)){
                hi = mid - 1;
                ans = mid;
                for(i = 0;i<m;i++){
                    s[i] = flow[x[i]][y[i]]+c[i]*f[i];
                }
            }
            else low = mid+1;
        }
        printf("%d\n",ans);
        for(i = 0;i<m;i++){
            if(i) printf(" ");
            printf("%d",s[i]);
        }
        puts("");
    }
    return 0;
}
```

8-Queen Problem:

The problem can be quite computationally expensive as there are 4,426,165,368 (ie., 64 choose 8) possible arrangements of eight queens on a board, but only 92 solutions. It is possible to use shortcuts that reduce computational requirements or rules of thumb that avoids brute force computational techniques. For example, just by applying a simple rule that constrains each queen to a single column (or row), though still considered brute force, it is possible to reduce the number of possibilities to just 16,777,216 (that is, 88) possible combinations. Generating the permutations that are solutions of the eight rooks puzzle[1] and then checking for diagonal attacks further reduces the possibilities to just 40,320 (that is, 8!). These are computationally manageable for n = 8, but would be intractable for problems of n = 20, as 20! = 2.433 * 1018. Extremely interesting advancements for this and other toy problems is the development and application of heuristics (rules of thumb) that yield solutions to the n queens puzzle at an astounding fraction of the computational requirements. This heuristic solves n queens for any n n = 4 or n = 1:
Divide n by 12. Remember the remainder (n is 8 for the eight queens puzzle).
Write a list of the even numbers from 2 to n in order.
If the remainder is 3 or 9, move 2 to the end of the list.
Append the odd numbers from 1 to n in order, but, if the remainder is 8, switch pairs (i.e. 3, 1, 7, 5, 11, 9, …).
If the remainder is 2, switch the places of 1 and 3, then move 5 to the end of the list.
If the remainder is 3 or 9, move 1 and 3 to the end of the list.
Place the first-column queen in the row with the first number in the list, place the second-column queen in the row with the second number in the list, etc.
For n = 8 this results in the solution shown above. A few more examples follow.
14 queens (remainder 2): 2, 4, 6, 8, 10, 12, 14, 3, 1, 7, 9, 11, 13, 5.
15 queens (remainder 3): 4, 6, 8, 10, 12, 14, 2, 5, 7, 9, 11, 13, 15, 1, 3.
20 queens (remainder 8): 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 1, 7, 5, 11, 9, 15, 13, 19, 17.

Bipartite Matching:

```
vector<int> wom[MX];
int visited[MX], hs[MX], wf[MX];
bool tri(int m){
    if(visited[m])  return false;
    visited[m] = true;
    int i;
    FOR(i,wom[m].size())
        if(hs[wom[m][i]]==-1 || tri(hs[wom[m][i]])){
            hs[wom[m][i]] = m, wf[m] = wom[m][i];
            return true;
        }
    return false;
}
int BPM(int man_size){
    int i,ret = 0;
    MCLR(hs), MCLR(wf), CLR(visited);
    FOR(i,man_size) if(wf[i]== -1){
            CLR(visited);
            if(tri(i))  ret++;
        }
    return ret;
}
```

Gomory Hu Tree (All Pair Max Flow) O(V * MaxFlow):

```
int cap[205][205],flow[205][205],prev[205],w[205],p[205],n;
bool bfs(int s,int t){
    int u,i;
    memo(prev,-1);
    queue<int> q;
    q.push(s);
    prev[s]=-2;
    while(!q.empty()){
        u = q.front();
        q.pop();
        for(i=0;i<n;i++){
            if(prev[i]==-1 && cap[u][i]-flow[u][i]>0){
                prev[i]= u;
                q.push(i);
            }
        }
    }
    if(prev[t]==-1) return false;
    return true;
}
int dinic(int s,int t){
    memo(flow,0);
    int tot = 0,fl,u,v,i;
    while(bfs(s,t)){
        for(i=0;i<n;i++){
            if(prev[i]==-1 || cap[i][t]-flow[i][t]<=0) continue;
            fl = cap[i][t] - flow[i][t];
            for(v = i,u=prev[v];v>=0;v =u,u=prev[v]) fl = min(fl,cap[u][v]-flow[u][v]);
            for(v = i,u=prev[v];v>=0;v =u,u=prev[v]) {
                flow[u][v]+=fl,flow[v][u]-=fl;
            }
            flow[i][t]+=fl;
            flow[t][i]-=fl;
            tot+=fl;
        }
    }
    return tot;
}
```

```
int main(){
    int t,cs,i,j,source,sink,fl,k,inf=1<<29;
    scanf("%d",&t);
    for(cs=1;cs<=t;cs++){
        scanf("%d",&n);
        for(i=0;i<n;i++){
            p[i]=0;
            for(j=0;j<n;j++) scanf("%d",&cap[i][j]);
        }
        for(source = 1;source<n;source++){
            sink = p[source];
            fl = dinic(source,sink);
            for(i=0;i<n;i++){
                if(i==source || prev[i]==-1 || p[i]!=sink) continue;
                p[i]=source;
            }
            w[source]=fl;
            if(prev[p[sink]]==-1) continue;
            p[source]=p[sink];
            p[sink]=source;
            w[source]=w[sink];
            w[sink]=fl;
        }
        for(i=0;i<n;i++) for(j=0;j<n;j++) cap[i][j]=0;
        for(i=0;i<n;i++){
            if(p[i]==i) continue;
            cap[i][p[i]]=cap[p[i]][i]=w[i];
        }
        for(k = 0;k<n;k++)
            for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                    cap[i][j]=max(cap[i][j],min(cap[i][k],cap[k][j]));
        for(i=0;i<n;i++) cap[i][i]=0;
        printf("Case #%d:\n",cs);
        for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                if(j) printf(" ");
                printf("%d",cap[i][j]);
            }
            puts("");
        }
    }
    return 0;
}
```

LIS O(n log n) with full path:

```
int num[MX], mem[MX], prev[MX], array[MX], res[MX],maxlen;
void LIS(int SZ,int num[]){
    CLR(mem), CLR(prev), CLR(array), CLR(res) ;
    int i , k;
    maxlen = 1;
    array[0] = -inf;
    RFOR(i,1,SZ+1)  array[i] = inf;
    prev[0] = -1, mem[0] = num[0];
    FOR(i,SZ){
        k = lower_bound( array , array + maxlen + 1, num[i] ) - array;
        if( k == 1) array[k] = num[i], mem[k] = i, prev[i] = -1;
        else    array[k] = num[i], mem[k] = i,prev[i] = mem[k -1];
        if(k > maxlen)  maxlen = k;
    }
    k = 0;
    for(i = mem[maxlen];i != -1;i = prev[i])    res[k++] = num[i];
}
```

Matrix Multiplication:

```
struct matrix{
    int n[SZ][SZ];
};
matrix iden;    // put 1 in the diagonal of the identity matrix
int M;          // the MOD value
matrix mul(matrix &a,matrix &b){
    matrix c;
    int i,j,t,k;
    for(i = 0;i<SZ;i++) for(j = 0;j<SZ;j++){
            t = 0;
            for(k = 0;k<SZ;k++)     t=(t+(a.n[i][k]*b.n[k][j]) %M)%M;
            c.n[i][j] = t;
        }
    return c;
}
matrix raise(matrix &m,i64 p)
{
    matrix tmp = iden;
    int r = 0, c = 0;
    while(p) r |= p&1, p = p>>1,    c++, r = r<<1;
    for(r = r >> 1;c;r = r>>1,c--){
        tmp = mul(tmp,tmp);
        if( r &1 ) tmp = mul(tmp,m);
    }
    return tmp;
}
```

Minimal enclosing circle given some points:

```
// Requires: ConvexHull(), circle CircleThrough3Points
struct Point{
    LD x,y;
    Point(){}
    Point(double a, double b){x = a, y = b;}
    bool operator <(Point b)const{
        if(!eq(x,b.x) ) return x < b.x;
        return y < b.y;
    }
    bool operator == (Point b) const{
        if(eq(x,b.x) && eq(y,b.y))  return true;
        return false;
    }
};
double ang(Point a,Point b,Point c){    //returns angle <bac
    double absq = sq_distance(a , b);
    double bcsq = sq_distance(c , b), acsq = sq_distance(a , c);
    double cosp = (absq+acsq - bcsq)/(2.0*sqrt(absq * acsq) );
    return acos(cosp);
}
struct side{
    Point a,b;
    side(){}
    side(Point aa,Point bb){ a = aa,b = bb;}
};
struct circle{
    Point center;
    double r;
};
```

```
int main(){
    int tc, i;
    cin>>tc;
    while(tc--){
        P.clear();
        int n;
        double a,b;
        cin >> n;
        while(n -- )
            scanf("%lf %lf",&a,&b), P.push_back(Point(a,b));
        ConvexHull();
        circle res;
        side now(C[0],C[1]);

        int ai = 0, aj = 1;
        while(true) {
            double tmp, mn = 1e10;
            int rem;
            FOR(i,nC){
                if(i!= ai && i!=aj){
                    tmp = ang(C[i],C[ai],C[aj]);
                    if(tmp < mn) mn = tmp, rem = i;
                }
            }
            if( 2*mn >= PI){
                res.r = sqrt(sq_distance(C[ai], C[aj]))/2.0;
                res.center.x = (C[ai].x + C[aj].x)/2;
                res.center.y = (C[ai].y + C[aj].y)/2;
                break;
            }
            double a1 = ang(C[ai],C[aj],C[rem]);
            double a2 = ang(C[aj],C[ai],C[rem]);
            double a3 = ang(C[rem],C[ai],C[aj]);
            if(a1 <= RA && a2 <=RA && a3 <= RA) {
                res = CircleThrough3Points(C[ai],C[aj],C[rem]);
                break;
            }
            else if(a1 > RA ) ai = aj, aj = rem;
            else if(a2 > RA ) ai = ai, aj = rem;
        }
        printf("%.2lf\n%.2lf %.2lf\n",res.r,res.center.x,res.center.y);
    }
    return 0;
}
```

Lenguar Tarjan algorithm:

```
#define MAX 5005
vector<int> adj[MAX],pred[MAX],bucket[MAX];
bool vi[MAX];
int n,cnt,num[MAX],par[MAX],ancestor[MAX],best[MAX],semi[MAX],idom[MAX],rnum[MAX];

void dfs(int x,int p){
    vi[x] = 1;
    num[x] = cnt++;
    rnum[cnt-1] = x;
    par[num[x]] = num[p];
    int sz = adj[x].size(),i,y;
    for(i = 0;i<sz;i++){
        y = adj[x][i];
        if(!vi[y]){
            dfs(y,x);
        }
    }
}
```

```
void compress(int x){
    int a = ancestor[x];
    if(ancestor[a]==0) return;
    compress(a);
    if(semi[best[x]] > semi[best[a]])
        best[x] = best[a];
    ancestor[x] = ancestor[a];
}
int eval(int x){
    if(ancestor[x]==0) return x;
    compress(x);
    return best[x];
}
int main(){
    int i,x,y,m,j,p;
    while(scanf("%d %d",&n,&m)==2){
        for(i = 1;i<=n;i++){
            adj[i].clear(),pred[i].clear(),bucket[i].clear();
            ancestor[i] = idom[i] = vi[i] = 0;
            semi[i] = best[i] = i;
        }
        for(i = 0;i<m;i++){
            scanf("%d %d",&x,&y);
            adj[x].pb(y);
            pred[y].pb(x);
        }
        cnt = 1,num[0] = 0;
        dfs(1,0);

        set<int> s;
        set<int>::iterator it;
        for(i = n;i>1;i--){
            p = par[i];
            int sz = pred[rnum[i]].size();
            for(j = 0;j<sz;j++){
                x = num[pred[rnum[i]][j]];
                y = eval(x);
                if(semi[i]>semi[y])
                    semi[i] = semi[y];
            }
            bucket[semi[i]].pb(i);
            ancestor[i] = p;       // link

            sz = bucket[p].size();
            for(j = 0;j<sz;j++){
                x = bucket[p][j];
                y = eval(x);
                if(semi[y]<p) idom[x] = y;
                else idom[x] = p;

            }
        }

        s.insert(1);
        for(i = 2;i<=n;i++){
            if(idom[i]!=semi[i])
                idom[i]=idom[idom[i]];
            s.insert(rnum[idom[i]]);
        }
        it = s.begin();
        printf("%d\n%d",s.size(),*it);
        for(it++;it!=s.end();it++) printf(" %d",*it);
        puts("");
    }
    return 0;
}
```

Circle Union Area - O(n3logn):

```
// slicing + interval mainpulation, n^2 * nlogn = n^3logn

#define MAX 102
#define EPS 1e-9
double pi = acos(-1.);
struct Circle{
    double x,y,r,xlo,xhi;
    Circle(){}
    Circle(double a, double b, double c){x = a, y = b, r = c; xlo=x-r; xhi=x+r; }
}tmp;
int n;
Circle c[MAX];
#define OPEN  0
#define CLOSE 1
struct Event{
    int type;
    double y1,y2,aa; //aa = arcarea of the event's host circle
    Event(){}
    Event(int t,double yy1,double yy2,double aaa){
        type = t, y1 = yy1, y2 = yy2, aa = aaa;
    }
};
bool operator <(const Event &p,const Event &q){ //event sort function
    double py = p.y1 + p.y2, qy = q.y1 + q.y2;
    if(fabs(py-qy) < EPS) return p.type < q.type;
    return py > qy + EPS;
}
//this is enough as, p.y1 > q.y1 <=> p.y2 > q.y2 (slicing)
//circles MUST intersect. returns only the 'x' of intersections

double mysqrt(double s){
    if(s<EPS) return 0;
    return sqrt(s);
}
double D1( Circle &c1, Circle &c2 ){
    return mysqrt( ( c1.x - c2.x ) * ( c1.x - c2.x ) + ( c1.y - c2.y ) * ( c1.y - c2.y ) );
}
double D2( Circle &c1, Circle &c2 ){
    return ( ( c1.x - c2.x ) * ( c1.x - c2.x ) + ( c1.y - c2.y ) * ( c1.y - c2.y ) );
}
double S( double a ) { return a * a ; }
bool getCircleIsects(Circle c1,Circle c2,double &x1,double &x2){
    if(c1.r +EPS< c2.r)swap(c1,c2);
    double d = D1(c1,c2);
    double a = ( S(c1.r) + S(d) - S(c2.r) )/(2.*d);
    double h = mysqrt( S(c1.r) - S(a) );
    x1 = c1.x + (a*(c2.x-c1.x) - h*(c2.y-c1.y))/d;
    x2 = c1.x + (a*(c2.x-c1.x) + h*(c2.y-c1.y))/d;
    return true;
}
double res;
vector<double> vx, X; //for slicing
int ne, tos;
Event e[2*MAX]; //events of arc open/close
Event Stack[MAX+1]; //Stack - remember earlier openings
double inarea[MAX+1];//Stack - calculate the area that is covered by inner circles

double myacos( double a ){
    if( fabs( a + 1 ) < EPS ) return pi;
    if( fabs( a - 1 ) < EPS ) return 0;
    return acos( a );
}
```

```
double arcarea( double C, double D, Circle &c ){
    double AOC, BOD, AOB, O = c.x;
    if( C > O + EPS && D > O + EPS ) swap( C, D );
    double OC = fabs( O - C ), OD = fabs( O - D );
    AOC = myacos( OC / c.r ), BOD = myacos( OD / c.r );
    if(( C + EPS < O && D + EPS < O ) || ( C > O + EPS && D > O + EPS ) );
    else BOD = pi - BOD;
    AOB = ( BOD - AOC ) / 2;
    return c.r * c.r *( AOB - cos(AOB)*sin(AOB) );
}
bool comp(const double &aa,const double &bb){
    return aa+EPS < bb;
}
void circleUnionArea(int n,Circle *c){
    int i,j,k;
    double d,x1,x2;
    // << slicing starts >>
    vx.clear();
    for(i=0;i<n;i++){ //no need for center.x
        vx.push_back(c[i].x - c[i].r);
        vx.push_back(c[i].x + c[i].r);
    }
    //insert all possible x[intersections]
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++){
        d = D1(c[i],c[j]);
        double dd2 = D2(c[i],c[j]);
        double ss1 = S(c[i].r + c[j].r);
        double ss2 = S(c[i].r - c[j].r);
        if(dd2 > ss1 + EPS||fabs(dd2 - ss1)<EPS || dd2 + EPS < ss2 || fabs(dd2 - ss2)<EPS)
            continue;
        if( getCircleIsects(c[i],c[j],x1,x2) )  vx.push_back(x1), vx.push_back(x2);
    }
    sort(vx.begin(), vx.end(),comp); X.clear(); X.push_back(vx[0]);
    for(i=1;i<vx.size();i++)
        if( fabs(vx[ i - 1 ] - vx[ i ] ) > EPS ) X.push_back(vx[i]);
    // << slcing end >>
    double area,xgap, OC, AC, OD,BD;
    for(k=0;k+1<X.size();k++){ //for each X slice
        x1 = X[k]; x2 = X[k+1]; xgap = x2-x1; ne = 0;
        for(i=0;i<n;i++){ // 2 events for each circle
            if(x2 + EPS < c[i].xlo || fabs(x2 - c[i].xlo)<EPS) continue;
            if(c[i].xhi +EPS < x1  || fabs(x1 - c[i].xhi)<EPS) continue;
            OC = (x1-c[i].x);
            AC = mysqrt( S(c[i].r) - S(OC) );
            OD = (x2-c[i].x);
            BD = mysqrt( S(c[i].r) - S(OD) );
            area = arcarea(x1,x2,c[i]);
            e[ne++] = Event(OPEN, c[i].y+AC, c[i].y+BD, area);
            e[ne++] = Event(CLOSE,c[i].y-AC, c[i].y-BD, area);
        }
        if(ne==0)continue;
        sort(e,e+ne);
        for(i=0;i<=n;i++)inarea[i] = 0; //init the inner area sum
        tos = 0;
        for(i=0;i<ne;i++)
            if(e[i].type == CLOSE){
                area = Stack[tos-1].aa + e[i].aa;
                area += 0.5*xgap*((Stack[tos-1].y1 - e[i].y1)+(Stack[tos-1].y2 - e[i].y2));
                res += area - inarea[tos - 1];
                if(tos>=2)  inarea[tos-2] += area;
                inarea[tos-1] = 0; tos--;
            }
            else Stack[tos++] = e[i];
    }
}
```

```
int main(){
    int i, CC;
    while( scanf("%d",&n) == 1 && n){
        CC = 0;
        for(i = 0; i < n ; i ++ ){
            scanf("%lf %lf %lf",&tmp.x,&tmp.y,&tmp.r);
            if(tmp.r<EPS) continue;
            c[ CC ++ ] =  Circle(tmp.x,tmp.y,tmp.r );
        }
        res = 0;
        circleUnionArea( CC, c );
        printf("%.3lf\n",res + EPS );
    }
    return 0;
}
```

Maximum matching on a general graph using Edmond's Blossom Shrinking O(N^3):

```
#define VI vector<int>
#define VVI vector< VI >

VVI adj;
VI vis,inactive,match;
int N;
bool dfs(int x,VI &blossom){
    if(inactive[x]) return false;
    int i,y;
    vis[x] = 0;
    for(i = adj[x].size()-1;i>=0;i--){
        y = adj[x][i];
        if(inactive[y]) continue;
        if(vis[y]==-1){
            vis[y] = 1;
            if(match[y]==-1 || dfs(match[y],blossom)){
                match[y] = x;
                match[x] = y;
                return true;
            }
        }
        if(vis[y]==0 || blossom.size()){
            blossom.push_back(y);
            blossom.push_back(x);
            if(blossom[0]==x){
                match[x] = -1;
                return true;
            }
            return false;
        }
    }
    return false;
}
bool augment(){
    VI blossom,mark;

    int i,j,k,s,x;
    for(i = 0;i<N;i++){
        if(match[i]!=-1) continue;
        blossom.clear();
        vis = VI(N+1,-1);
        if(!dfs(i,blossom)) continue;
        s = blossom.size();
        if(s==0) return true;


        mark = VI(N+1,-1);
```

```
        for(j = 0;j<s-1;j++){
            for(k = adj[blossom[j]].size()-1;k>=0;k--) mark[adj[blossom[j]][k]] = j;
        }
        for(j = 0;j<s-1;j++){
            mark[blossom[j]] = -1;
            inactive[blossom[j]] = 1;
        }
        adj[N].clear();
        for(j = 0;j<N;j++){
            if(mark[j]!=-1)
                adj[N].pb(j),adj[j].pb(N);
        }
        match[N] = -1;
        N++;
        if(!augment()) return false;
        N--;

        for(j = 0;j<N;j++){
            if(mark[j]!=-1) adj[j].pop_back();
        }
        for(j = 0;j<s-1;j++){
            inactive[blossom[j]] = 0;
        }
        x = match[N];
        if(x!=-1){
            if(mark[x]!=-1){
                j = mark[x];
                match[blossom[j]] = x;
                match[x] = blossom[j];
                if(j & 1)
                    for(k = j+1;k<s;k+=2) {
                        match[blossom[k]] = blossom[k+1];
                        match[blossom[k+1]] = blossom[k];
                    }
                else
                    for(k = 0;k<j;k+=2) {
                        match[blossom[k]] = blossom[k+1];
                        match[blossom[k+1]] = blossom[k];
                    }
            }
        }

        return true;
    }
    return false;
}
int main(){
    int i,x,y,m,ret;
    while(scanf("%d",&N)==1 && N){
        scanf("%d",&m);
        adj = VVI(2*N+1);

        for(i = 0;i<m;i++){
            scanf("%d %d",&x,&y);
            adj[x].pb(y);
            adj[y].pb(x);
        }
        match = VI(2*N+1,-1);
        inactive = VI(2*N+1);
        ret = 0;
        while(augment()) ret++;
        cout<<ret<<endl;
        for(i = 0;i<N;i++) cout<<i<<" "<<match[i]<<endl;
    }
    return 0;
}
```

Gaussian Elimination (Modular):

```
vector <int> a[ MX ], b;    // a contains co factors, b contains right side
int mi[] = {}               // contains modular inverses
int gauss (){
    // m = number of equations
    // n = number of variables
    // MOD = the number to take MOD

      int ii = 0 , i, j;
      for(i = 0; i < n ; i ++ ) {
        j = ii ;
        while ( j < m && a[j][i] == 0 )  j++ ;
        if (j == m)  continue ;

        swap (a[ii], a[j]) ;
        swap (b[ii], b[j]) ;

        int t = a[ii][i] , k;
        for(k = 0; k < n ; k ++ )  a[ii][k] = ((a[ii][k] * mi[ t ])%MOD+MOD)%MOD;
        b[ii] = ((b[ii] *  mi[ t ])%MOD+MOD)%MOD;

        for(j = 0; j < m ; j ++ )  if (j!=ii) {
            int t = a[j][i] ;
            for(k = 0; k < n ; k ++ )   a[j][k] = ( (a[j][k] - t*a[ii][k])%MOD+MOD)%MOD;
            b[j] = ( (b[j] - t*b[ii])%MOD+MOD)%MOD;
          }
        ii++ ;
    }
    for (i = ii; i <= m-1; i ++) if (b[i]!=0) return -1 ;   // inconsistent
    if (ii<n) return 1 ;    // multiple solutions

     return 0 ;
}
```

Maximum point cover with circle of radius R:

```
point P[2010];
bool invalid(int i,int j,double r){if( i == j ) return true;    if( sqdist(P[ i ],P[ j ]) >
eps + double (4*r*r) ) return true; return false;}
double get_angle(point a, point b){
    if( a.x == b.x && b.y > a.y ) return PI/2;
    if( a.x == b.x && b.y < a.y ) return 3*PI/2;
    if( a.y == b.y && b.x > a.x ) return 0;
    if( a.y == b.y && b.x < a.x ) return PI;

    int dy = b.y - a.y, dx = b.x - a.x;
    return atan2((double)dy,(double)dx);
}
typedef pair <double, int> nd;
typedef pair <nd, int> node;
node V[ 4010 ];
bool bhitre( point a, point b, int r ){
    int s = ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
    if( s <= r * r ) return true;
    return false;
}
```

```
int main(){
    int n, i, j, k, r, x, y;
    while( cin >> n >> r && (n+r) ){
        FOR(i,n) scanf("%d %d",&x, &y ), P[ i ] = point(x,y);
        int res = 0, cnt;
        FOR(i,n){
            k = 0, cnt = 0;
            FOR(j,n){
                if( invalid(i,j,r) ) continue;
                double ds = sqrt( sqdist(P[ i ],P[ j ]) );
                double ang = get_angle(P[ i ], P[ j ] );
                double ang1 = ang - acos(ds / ( 2.*(double)r ) ) - eps/10;
                double ang2 = ang + acos(ds / ( 2.*(double)r ) ) + eps/10;

                while( ang1 < 0 ) ang1 += 2*PI;
                while( ang1 >= 2*PI ) ang1 -= 2*PI;
                while( ang2 < 0 ) ang2 += 2*PI;
                while( ang2 >= 2*PI ) ang2 -= 2*PI;

                V[ k ++ ] = node( nd(ang1,-1), j);
                V[ k ++ ] = node( nd(ang2 ,1), j);
            }
            int cnt = 0;
            bool fl[ 2010 ] = { 0 };
            FOR(j,n) if( bhitre( point( P[ i ].x + r, P[ i ].y) , P[ j ], r ) )
                            cnt --, fl[ j ] = 1;
            sort(V, V + k );
            int sz = k;
            if( -cnt > res ) res = -cnt;
            for(j = 0; j < sz; j ++ ){
                if( fl[ V[ j ].second ] && V[ j ].first.second == -1) ;
                            //bhitre ase and ekhon abar dhukaite chai, ignore
                else if( !fl[ V[ j ].second ] && V[ j ].first.second == 1) ;
                            //bhitre nai and ekhon abar ber korte chai, ignore
                else cnt += V[ j ].first.second;                                    // normal
                if( V[ j ].first.second == 1) fl[ V[ j ].second ] = 0;
                            // bair korle bhitre ase er flag off
                if( V[ j ].first.second == -1) fl[ V[ j ].second ] = 1;
                        // dhukaile bhitre ase er flag on
                if( -cnt > res ) res = -cnt;
            }
        }
        printf("It is possible to cover %d points.\n",res);
    }
    return 0;
}
```

Sudoku with DLX Algorithm:

```
#define MAX_ROW 900
#define MAX_COL 400

struct node {
    node *Header;
    node *Left,*Right;
    node *Up,*Down;
    int id;
};
node Matrix[MAX_ROW][MAX_COL];
node *RowHeader[MAX_ROW];

node Root;
node *RootNode = &Root;

int nCol,nRow,nResult,n = 9, Solved, m = sqrt(n),cnt;
```

```
int Result[MAX_ROW],data[MAX_ROW][MAX_COL];

inline int dataLeft(int i){
    return i-1<0?nCol - 1:i-1;
}
inline int dataRight(int i){
    return (i+1) % nCol;
}
inline int dataUp(int i){
    return i-1<0?nRow - 1:i-1;
}
inline int dataDown(int i){
    return (i+1) % nRow;
}
void Build(){
    int i,j,a,b;
    for(i = 0;i<nRow;i++){
        for(j = 0;j<nCol;j++){
            if(data[i][j]==0) continue;
            a = i; b = j ;
            do{
                b = dataLeft(b);
            }while(data[a][b]==0);
            Matrix[i][j].Left = &Matrix[a][b];
            a = i; b = j ;
            do{
                b = dataRight(b);
            }while(data[a][b]==0);
            Matrix[i][j].Right = &Matrix[a][b];
            a = i; b = j ;
            do{
                a = dataUp(a);
            }while(data[a][b]==0);
            Matrix[i][j].Up = &Matrix[a][b];
            a = i; b = j ;
            do{
                a = dataDown(a);
            }while(data[a][b]==0);
            Matrix[i][j].Down = &Matrix[a][b];
            Matrix[i][j].Header = &Matrix[nRow - 1][j];
            Matrix[i][j].id  = i;
            RowHeader[i] = &Matrix[i][j];
        }
    }
    for(i = 0;i<nCol;i++) Matrix[nRow-1][i].id = i;
    Root.Left = &Matrix[nRow-1][nCol-1];
    Root.Right = &Matrix[nRow-1][0];
    Matrix[nRow-1][0].Left = &Root;
    Matrix[nRow - 1][nCol - 1].Right = &Root;
}
void printSol(){
    int i,j;
    char tmp[20][20];
    for(i = 0;i<nResult;i++){
        tmp[(Result[i]/n)%n][Result[i]%n] = Result[i] / (n*n) + 49;
    }
    for(i = 0;i<n;i++){
        for(j = 0;j<n;j++){
            if(j) printf(" ");
            printf("%c",tmp[i][j]);
        }
        puts("");
    }

}
```

```
void cover(node *col){
    node *rowNode,*rightNode;
    col->Right->Left = col->Left;
    col->Left->Right = col->Right;
    for(rowNode = col->Down;rowNode!=col;rowNode = rowNode->Down){
        for(rightNode = rowNode->Right;rightNode!=rowNode;rightNode = rightNode->Right){
            rightNode->Down->Up = rightNode->Up;
            rightNode->Up->Down = rightNode->Down;
        }
    }
}

void uncover(node *col){
    node *rowNode,*leftNode;

    for(rowNode = col->Up;rowNode!=col;rowNode = rowNode->Up){
        for(leftNode = rowNode->Left;leftNode!=rowNode;leftNode = leftNode->Left){
            leftNode->Down->Up = leftNode;
            leftNode->Up->Down = leftNode;
        }
    }
    col->Left->Right = col;
    col->Right->Left = col;
}
void search(int depth){
    if(RootNode->Right==RootNode && RootNode->Left==RootNode || depth == n*n - cnt) {
        printSol();
        Solved ++;
        return;
    }
    node *col = RootNode->Right;
    cover(col);
    node *rowNode,*rightNode;

    for(rowNode = col->Down;rowNode!=col && !Solved;rowNode = rowNode->Down){

        Result[nResult++] = rowNode->id;
        for(rightNode = rowNode->Right;rightNode!=rowNode;rightNode = rightNode->Right){
            cover(rightNode->Header);
        }
        search(depth + 1);
        for(rightNode = rowNode->Right;rightNode!=rowNode;rightNode = rightNode->Right){
            uncover(rightNode->Header);
        }
        Result[nResult--] = 0;
    }
    uncover(col);
}
inline int getRow(int i,int j,int k){
    return i*n*n + j*n + k;
}
inline int getSq(int i,int j){
    return i*n + j;
}
inline int getRw(int i,int j){
    return n*n + i*n+j;
}
inline int getCl(int i,int j){
    return 2*n*n + i*n+j;
}
inline int getBx(int i,int j,int k){
    return 3*n*n + ((j/m) * m + k/m)*n+i;
}
```

```
void createData(){
    nRow = n*n*n + 1;
    nCol = 4*n*n;
    int i,j,k;
    for(i = 0 ;i<n;i++){
        for(j = 0;j<n;j++){
            for(k = 0;k<n;k++){
                int indx = getRow(i,j,k);
                data[indx][getSq(j,k)] = 1;
                data[indx][getRw(i,j)] = 1;
                data[indx][getCl(i,k)] = 1;
                data[indx][getBx(i,j,k)] = 1;
            }
        }
    }
    for(i = 0;i<nCol;i++) data[nRow - 1][i] = 1;
}

void add(int i,int j,int k){
    int idx = getRow(i,j,k);
    cover(RowHeader[idx]->Header);
    node *rightNode;
    for(rightNode = RowHeader[idx]->Right;rightNode!=RowHeader[idx];rightNode = rightNode-
>Right){
        cover(rightNode->Header);
    }
    cnt++;
    Result[nResult++] = idx;
}
void readInput(){
    int i,j;
    int tmp;
    for(i = 0;i<n;i++){

        for(j = 0;j<n;j++){
            scanf("%d",&tmp);
            if(tmp==0) continue;
            add(tmp - 1,i,j);
        }
    }

}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        memo(data,0);
        cnt = 0;
        createData();
        Build();
        readInput();

        Solved = 0;
        search(0);
        if(!Solved) puts("No solution");

    }
    return 0;
}
```

Bipartite Matching Using Hopcroft Carp:

```
int d[105],rPair[105],lPair[105],inf=1<<29,m,n;
bool gr[105][105],vi[105];
bool BFS(){
    d[0]=inf;
    queue<int> q;
    int i,v;
    for(i=1;i<=n;i++){
        if(rPair[i]==0){
            d[i]=0;
            q.push(i);
        }
        else    d[i]=inf;
    }
    while(!q.empty()){
        v = q.front();
        q.pop();
        if(v == 0) continue;
        for(i = 1 ; i<=m;i++){
            if(!gr[v][i]) continue;
            if(d[lPair[i]]==inf){
                d[lPair[i]]= d[v]+1;
                q.push(lPair[i]);
            }
        }
    }
    return d[0]!=inf;
}
bool DFS(int v){
    if(vi[v]) return 0;
    vi[v]=1;
    if(v==0) return 1;
    int u;
    for(u=1;u<=m;u++){
        if(!gr[v][u]) continue;
        if(d[lPair[u]]==d[v]+1){
            if(DFS(lPair[u])){
                lPair[u]=v, rPair[v]=u;
                return 1;
            }
        }
    }
    d[v]=inf;
    return 0;
}
int main(){
    int ne,i,x,y,match;
    while(scanf("%d %d %d",&n,&m,&ne)==3){
        memo(gr,0);
        for(i=0;i<ne;i++){
            scanf("%d %d",&x,&y);
            gr[x][y]=1;
        }
        memo(rPair,0),  memo(lPair,0);
        match=0;
        while(BFS())
            for(i=1;i<=n;i++){
                if(rPair[i]==0){
                    memo(vi,0);
                    if(DFS(i)) match++;
                }
            }
        printf("%d\n",match);
    }
    return 0;
}
```

Longest Palindrome O(n):

```
int main(){
    string s;
    while(cin>>s){
        int n = s.length();
        int i = 0,pal = 0;
        vector<int> v;
        while(i<n){
            while(i>pal && s[i]==s[i - pal - 1]){
                pal+=2;
                i++;
            }
            v.pb(pal);
            int s = v.size() - 2;
            int e = s - pal, j ;
            for(j = s;j>e;j--){
                int d = j - e - 1;
                if(v[j]==d){
                    pal = d;
                    break;
                }
                v.pb(min(d,v[j]));
            }
            if(j==e){
                pal = 1;
                i++;
            }
        }
        v.pb(pal);
        int len = v.size();
        int s = len - 2;
        int e = s - (2*n + 1 - len);
        for(i = s;i>e;i--){
            int d = i - e -1;
            v.pb(min(d,v[i]));
        }
        len = 0;
        for(i = v.size() - 1;i>=0;i--) len = max(len,v[i]);
        cout<<len<<endl;
    }
    return 0;
}
```

Rectangle Union O(n log n):

```
struct rect{
    int lx,ly,rx,ry;
}R[10005];
struct Axes{
    int x,type,id;
    Axes(){}
    Axes(int xx,int tt,int idd){
        x = xx;
        type = tt;
        id = idd;
    }
    bool operator<(const Axes &a)const{
        if(x<a.x) return 1;
        if(x==a.x && type<a.type) return 1;
        return 0;
    }
}A[20005];

int T[30005*4],D[30005*4];
```

```
void insert(int lf,int rt,int id,int x,int y){
    if(lf>y || rt<x) return;
    if(lf>=x && rt<=y){
        if(T[id]==-1) T[id] = 1;
        else T[id]++;
        return;
    }
    if(T[id]!=-1){
        if(T[2*id]!=-1) T[2*id]+=T[id];
        else T[2*id]=T[id];
        if(T[2*id+1]==-1) T[2*id+1]+=T[id];
        else T[2*id+1] =T[id];
        T[id] = -1;
    }
    insert(lf,(lf+rt)/2,2*id,x,y);
    insert((lf+rt)/2+1,rt,2*id+1,x,y);
}
void remove(int lf,int rt,int id,int x,int y){
    if(lf>y || rt<x) return;
    if(lf>=x && rt<=y){
        if(D[id]==-1) D[id] = 1;
        else D[id]++;
        return;
    }
    if(D[id]!=-1){
        if(D[2*id]!=-1) D[2*id]+=D[id];
        else D[2*id]=D[id];
        if(D[2*id+1]==-1) D[2*id+1]+=D[id];
        else D[2*id+1] =D[id];
        D[id] = -1;
    }
    remove(lf,(lf+rt)/2,2*id,x,y);
    remove((lf+rt)/2+1,rt,2*id+1,x,y);
}
int getCount(int lf,int rt,int id,int len){
    if(T[id]!=-1 && D[id]!=-1){
        T[id]-=D[id];
        D[id] = 0;
        return (rt - lf+1) * len * (T[id]>0);
    }
    if(T[id]!=-1){
        if(T[2*id]!=-1) T[2*id]+=T[id];
        else T[2*id]=T[id];
        if(T[2*id+1]==-1) T[2*id+1]+=T[id];
        else T[2*id+1] =T[id];
        T[id] = -1;
    }
    if(D[id]!=-1){
        if(D[2*id]!=-1) D[2*id]+=D[id];
        else D[2*id]=D[id];
        if(D[2*id+1]==-1) D[2*id+1]+=D[id];
        else D[2*id+1] =D[id];
        D[id] = -1;
    }
    return getCount(lf,(lf+rt)/2,2*id,len) + getCount((lf+rt)/2+1,rt,2*id+1,len);
}
int main(){
    int n,i,mn = 1<<29,mx = 0,m;
    cin>>n;
    for(i = 0,m=0;i<n;i++){
        scanf("%d %d %d %d",&R[i].lx,&R[i].ly,&R[i].rx,&R[i].ry);

        A[m++] = Axes(R[i].lx,0,i);
        A[m++] = Axes(R[i].rx,1,i);
        R[i].rx--;
```

```
        R[i].ry--;
        mn = min(mn,R[i].ly);
        mx = max(mx,R[i].ry);
    }
    sort(A,A+m);
    T[1] = 0;
    D[1] = 0;
    int ans = 0;
    for(i = 0;i<m;i++){
        if(i)
            ans+=getCount(mn,mx,1,A[i].x - A[i-1].x);
        if(A[i].type==0){
            insert(mn,mx,1,R[A[i].id].ly,R[A[i].id].ry);
        }
        else {
            remove(mn,mx,1,R[A[i].id].ly,R[A[i].id].ry);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

2D LIS O(n log n):

```
typedef pair<int,int> pii;
pii p[100005];
set<pii> s[100005];
set<pii>::iterator it,it1;
int main(){
    int n,i,lo,hi,mid,lb,k,t,cs = 1;
    scanf("%d",&n);
    for(i = 0;i<n;i++) scanf("%d %d",&p[i].first,&p[i].second);
    s[0].insert(p[0]);
    k = 0;
    for(i = 1;i<n;i++){
        lo = 0; hi = k,lb = -1;
        while(lo<=hi){
            mid = (lo + hi) / 2;
            it = s[mid].lower_bound(p[i]);
            if(it!=s[mid].begin() ){
                it1 = it,it1--;
                if((*it1).first==p[i].first) it --;
            }
            if(it!=s[mid].begin() && (*(--it)).second<p[i].second)
                lo = mid + 1,lb = max(lb,mid);
            else hi = mid - 1;
        }
        lb++;
        k = max(k,lb);
        it = s[lb].lower_bound(pii(p[i].first,-inf));
        if(it==s[lb].end() || ((*it).first>p[i].first || (*it).second>p[i].second))
            s[lb].insert(p[i]);
        it = s[lb].upper_bound(p[i]);
        while(it!=s[lb].end()){
            if((*it).first>=p[i].first && (*it).second >= p[i].second){
                it1 = it, it1++;
                s[lb].erase(it);
                it = it1;
            }
            else break;
        }
    }
    printf("%d\n",k+1);
    return 0;
}
```

Determinant (Modulo):

```
int determinant(int mat[MAX][MAX],int n,int mod){
    int i,j,k,A[MAX][MAX];
    for(i = 0;i<n;i++)
        for(j = 0;j<n;j++) {
            A[i][j] = mat[i][j] %mod;
            if(A[i][j]<0) A[i][j]+=mod;
        }
    int res = 1;
    for(i = 0;i<n;i++){
        int j = i;
        while(j < n && A[j][i] == 0) j++;
        if(j == n) return 0;
        if(i!=j){
            for(k = 0;k<n;k++){
                int t = A[i][k];
                A[i][k] = A[j][k];
                A[j][k] = t;
            }
            res*=-1;
        }
        res = (res* A[i][i]) % mod;
        for(j = i + 1;j<n;j++){
            if(A[j][i]!=0){
                int fac = (A[j][i] * power(A[i][i],mod - 2,mod)) %mod;
                A[j][i] = 0;
                for(k = i + 1;k<n;k++){
                    A[j][k] = (A[j][k] - A[i][k] * fac) % mod;
                    if(A[j][k]<0) A[j][k]+=mod;
                }
            }
        }
    }
    if(res<0) res+=mod;
    return res;
}
```

Number of Minimum Spanning tree in a Graph :

```
#define MAX 105
struct edge{
    int x,y,cost;
    bool operator<(const edge &e)const{
        return cost<e.cost;
    }
}E[1005];
vector< vector<int> > adj;
int num[105],cnt,par[105],mat[105][105];
bool vi[105];
void dfs(int x){
    vi[x] = 1, num[x] = cnt++;
    int i,y,sz = adj[x].size();
    mat[num[x]][num[x]] = sz;
    for(i = 0;i<sz;i++){
        y = adj[x][i];
        if(!vi[y])  dfs(y);
        mat[num[x]][num[y]]  --;
    }
}
int calc(){
    int r1 = determinant(mat,cnt-1,3);
    int r2 = determinant(mat,cnt-1,10337);
    if(r2%3==r1) return r2;
    if((r2+10337)%3==r1) return r2 + 10337;
    if((r2+2*10337)%3==r1) return r2 + 2*10337;
    return 0;
}
```

```
int Find(int x){
    if(par[x]==x) return x;
    return par[x] = Find(par[x]);
}
int main(){
    int m,i,j,u,v,ret,mscnt,n;
    scanf("%d %d",&n,&m);
    for(i = 0;i<m;i++){
        scanf("%d %d %d",&E[i].x,&E[i].y,&E[i].cost);
    }
    sort(E,E+m);
    ret = 1, mscnt = 0;
    for(i = 1;i<=n;i++) par[i] = i;
    for(i = 0;i<m && mscnt<n - 1; i++){
        j = i;
        adj = vector< vector<int> > (n+1);
        while(j<m && E[i].cost == E[j].cost){

            u = Find(E[j].x);
            v = Find(E[j].y);
            if(u!=v){
                adj[u].pb(v);
                adj[v].pb(u);
            }
            j ++;
        }
        if(j>i+1){
            j = i ;
            memo(vi,0);
            while(j<m && E[i].cost == E[j].cost){
                u = Find(E[j].x);

                if(!vi[u]){
                    memo(mat,0);
                    cnt = 0;
                    dfs(u);
                    ret = (ret*calc())%31011;
                }
                j ++;
            }
        }
        j = i;
        while(j<m && E[i].cost == E[j].cost){

            u = Find(E[j].x);
            v = Find(E[j].y);
            if(u!=v){
                par[u] = par[v];
                mscnt++;
            }
            j ++;
        }
        i = j - 1;
    }
    printf("%d\n",ret);
    return 0;
}
```

Matrix Inverse (Modular) :

```
int main(){
    int n,i,j,det,k,l,p,q;
    int mat[10][10];
    scanf("%d",&n);
    for(i = 0;i<n;i++){
        for(j = 0;j<n;j++){
            scanf("%d",&mat[i][j]);
        }
    }
    int a = determinant(mat,n,2);
    int b = determinant(mat,n,13);
    int A[10][10],C[10][10],x,y;
    for(i = 0;i<n;i++){
        for(j = 0;j<n;j++){
            for(k = 0,p=0;k<n;k++){
                if(k == i) continue;
                for(l = 0,q = 0;l<n;l++){
                    if(l==j) continue;
                    A[p][q++] = mat[k][l];
                }
                p++;
            }
            x = determinant(A,n-1,2);
            x *= (i+j) & 1 ? -1:1;
            x%=2;
            if(x<0)x+=2;

            y = determinant(A,n-1,13);
            y *= (i+j) & 1 ? -1:1;
            y%=13;
            if(y<0) y+=13;
            y = (y*power(b,11,13))%13;
            if(y%2==x) C[j][i] = y;
            else if((y+13)%2==x) C[j][i] = y+13;
            else C[j][i] = -1;
        }
    }
    if(b % 2 == a) det = b;
    else if((b + 13) % 2 == a) det = b+13;
    else det = -1;
    printf("Det %d\n",det);
    for(i = 0;i<n;i++){
        for(j = 0;j<n;j++) printf(" %d",C[i][j]);
        puts("");
    }
    return 0;
}
```

Template:

Template:

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<cctype>

#include<cmath>
#include<iostream>
#include<fstream>

#include<string>
#include<vector>
#include<queue>
#include<map>
#include<algorithm>
#include<set>
#include<sstream>
#include<stack>
#include<utility>
#include<numeric>
#include<iterator>
using namespace std;
#pragma comment(linker,"/STACK:16777216")
#pragma warning(disable:4786)

#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))

#define FOR(i,n) for( i = 0 ; i<(n) ; i++)
#define RFOR(i,a,b)  for( i = (a) ; i<(b) ; i++)
#define CLR(a) memset(a,0,sizeof(a))
#define MCLR(a) memset(a,-1,sizeof(a))
#define SET(a) memset(a,-1,sizeof(a))
#define memo(a,b) memset(a,b,sizeof(a))

#define all(a) a.begin(),a.end()
#define pb push_back
#define mpp(a,b,c) make_pair(make_pair(a,b),c)

#define inf (1<<29)
#define i64 long long
#define pi (2.0*acos(0.0))
#define eps (1e-9)

typedef pair< int , int >  pii;

int main(){
    freopen(".txt","r",stdin);
    return 0;
}
```