```
/*********************
 * Convex Hull
 * ******************/
// tested on ACM problem 11065
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <vector>

using namespace std;

class Point {
public:
        int x, y;

        Point () {}
        Point (int _x, int _y) : x(_x), y(_y) {}
};

int ccw(const Point &a, const Point &b, const Point &c) {
        return a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
}

double dist(const Point &a, const Point &b) {
        return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}

class PointsCmp {
public:
        Point reference;

        bool operator () (const Point &a, const Point &b) {
                int t = ccw(reference, a, b);
                if (t != 0) return t > 0;
                return dist(reference, a) < dist(reference, b);
        }

        PointsCmp(const Point &_reference) : reference(_reference) {}
};

class Polygon {
public:
        vector <Point> points;

        Polygon convexHull() {
                Polygon tmp = *this;

                for (int i = 1; i < points.size(); ++i) {
                        if (tmp.points[i].y < tmp.points[0].y) {
                                swap(tmp.points[i], tmp.points[0]);
                        }
                }

                sort(tmp.points.begin()+1, tmp.points.end(), PointsCmp(tmp.points[0]));

                Polygon hull;

        if (tmp.size() < 3) {
            return hull;
        }

                hull.points.push_back(tmp.points[0]);
                hull.points.push_back(tmp.points[1]);
                hull.points.push_back(tmp.points[2]);

                int M = hull.points.size();

                for (int i = 3; i < points.size(); ++i) {
                        while (ccw(hull.points[M-2], hull.points[M-1], tmp.points[i]) < 0) {
                                hull.points.pop_back();
                                --M;
                        }
                        hull.points.push_back(tmp.points[i]);
                        ++M;
                }

                return hull;
        }

        double area() {
```

```
                int retval = 0.0;
                for (int i = 0; i < points.size(); ++i) {
                        retval += points[i].x * points[(i+1) % points.size()].y - points[(i+1) % points.size(
)].x * points[i].y;
                }

                return ((retval < 0) ? -retval : retval) / 2.0;
        }

        void output() {
                for (int i = 0; i < points.size(); ++i) {
                        printf("(%d, %d) ", points[i].x, points[i].y);
                }
                printf("\n");
        }
};

Polygon P;

bool load() {
        int n;
        scanf("%d", &n);
        P.points.resize(n);

        for (int i = 0; i < n; ++i) {
                scanf("%d%d", &P.points[i].x, &P.points[i].y);
        }

        return n;
}

/*********************
 *     Hungarian
 * ******************/
// tested on ACM ICPC live problem 3198
#include <cstdio>
#include <vector>
#include <algorithm>
#include <cmath>
#include <queue>

using namespace std;

#define MAX_R 100 // mora biti >= MAX_C
#define MAX_C 100
#define VELIKO 1000000

bool zero(int x) { return x == 0; }
bool zero(double x) {return fabs(x) < 1e-12; }

template <typename tip>
struct hungarian {
        int n, m;
        tip costs[MAX_R][MAX_C]; // pocente vrijednosti NE OSTAJU ocuvane
        bool ret[MAX_R][MAX_C]; // na kraju, jedinice su matching
        int stars;
        int star_r[MAX_R], star_c[MAX_C];
        int prime_r[MAX_R], prime_c[MAX_C];
        int cover_r[MAX_R], cover_c[MAX_C];

        void matching() {
                for ( ; n < m; ++n)
                        for (int c = 0; c < m; ++c)
                                costs[n][c] = 0;
                for (int r = 0; r < n; ++r) { star_r[r] = -1; cover_r[r] = 0; }
                for (int c = 0; c < m; ++c) { star_c[c] = -1; cover_c[c] = 0; }
                stars = 0;
                step1();
        }

        void step1() {
                for (int r = 0; r < n; ++r) {
                        tip mini = VELIKO;
                        for (int c = 0; c < m; ++c) mini = min(mini, costs[r][c]);
                        for (int c = 0; c < m; ++c) costs[r][c] -= mini;
                }
                step2();
        }

        void step2() {
```

```
                for (int r = 0; r < n; ++r) {
                        for (int c = 0; c < m; ++c) {
                                if (star_c[c] != -1) continue;
                                if (!zero(costs[r][c])) continue;
                                star_r[r] = c;
                                star_c[c] = r;
                                ++stars;
                                break;
                        }
                }
                step3();
        }

        void step3() {
                if (stars == m) {
                        for (int r = 0; r < n; ++r)
                                for (int c = 0; c < m; ++c)
                                        ret[r][c] = (star_r[r] == c);
                        return; // zavrsetak algoritma
                }
                for (int r = 0; r < n; ++r) cover_r[r] = 0;
                for (int c = 0; c < m; ++c) cover_c[c] = star_c[c] != -1;

                step4();
        }

        void step4() {
                queue <int> Q;
                for (int c = 0; c < m; ++c) if (!cover_c[c]) Q.push(c);

                for (; !Q.empty(); Q.pop()) {
                        int c = Q.front();
                        for (int r = 0; r < n; ++r) {
                                if (cover_r[r]) continue;
                                if (!zero(costs[r][c])) continue;
                                if (star_r[r] != -1) {
                                        cover_c[star_r[r]] = 0;
                                        cover_r[r] = 1;
                                        prime_r[r] = c;
                                        prime_c[c] = r;
                                        Q.push(star_r[r]);
                                } else {
                                        step5(r, c);
                                        return;
                                }
                        }
                }
                tip mini = VELIKO;
                for (int r = 0; r < n; ++r) {
                        if (!cover_r[r])
                                for (int c = 0; c < m; ++c)
                                        if (!cover_c[c])
                                                mini = min(mini, costs[r][c]);
                }
                step6(mini);
        }
        void step5(int r, int c) {
                while (star_c[c] != -1) {
                        int tmp_r = star_c[c];
                        star_r[r] = c;
                        star_c[c] = r;
                        c = prime_r[tmp_r];
                        r = tmp_r;
                }
                star_r[r] = c;
                star_c[c] = r;
                stars++;
                step3();
        }
        void step6(tip mini) {
                for (int r = 0; r < n; ++r)
                        for (int c = 0; c < m; ++c)
                                if (cover_r[r] && cover_c[c]) costs[r][c] += mini;
                                else if (!cover_r[r] && !cover_c[c]) costs[r][c] -= mini;
                step4();
        }
};

hungarian <int> H;
```

```
char ploca[100][100];
int N, M;

bool load() {
        scanf("%d%d", &N, &M);

        for (int i = 0; i < N; ++i) {
                scanf("%s", ploca[i]);
        }

        return N+M;
}

int my_abs(int x) { return x < 0 ? -x : x; }
int dist(pair <int, int> a, pair <int, int> b) {
        return my_abs(a.first - b.first) + my_abs(a.second - b.second);
}
vector <pair <int, int> > houses, men;

void generate_costs() {
        houses.clear(); men.clear();

        for (int i = 0; i < N; ++i) {
                for (int j = 0; j < M; ++j) {
                        if (ploca[i][j] == 'H') houses.push_back(make_pair(i, j));
                        if (ploca[i][j] == 'm') men.push_back(make_pair(i, j));
                }
        }

        H.n = H.m = houses.size();
        for (int i = 0; i < houses.size(); ++i) {
                for (int j = 0; j < men.size(); ++j) {
                        H.costs[i][j] = dist(houses[i], men[j]);
                }
        }
}


/*************************
 *     KMP
 * ************************/
// c/p from Zagreb
#define MAXP 1000
#define MAXT 1000

int pi[MAXP+1];
char T[MAXT+1]; int n;
char P[MAXP+1]; int m;

void compute_prefix_function() {
    pi[1] = 0;
    int k = 0;
    for (int q = 2; q <= m; ++q) {
        while (k > 0 && P[k] != P[q-1]) k = pi[k];
        if (P[k] == P[q-1]) ++k;
        pi[q] = k;
    }
}

void KMP_matcher() {
    int q = 0;

    for (int i = 1; i <= n; ++i) {
        while (q > 0 && P[q] != T[i-1]) q = pi[q];
        if (P[q] == T[i-1]) q++;
        if (q == m) {
            // we found pattern with shift i-m
            q = pi[q];
        }
    }
}


/*************************
 *     Matching
 * ************************/
#include <cstdio>
#include <vector>

using namespace std;
```

```
vector <vector <int> > E;
vector <int> connectedF, bio;

int dfs(int s) {
        if (bio[s]) return 0;
        bio[s] = 1;

        for (int i = 0; i < E[s].size(); ++i) {
                if (connectedF[E[s][i]] == s) continue;
                if (connectedF[E[s][i]] == -1 || dfs(connectedF[E[s][i]])) {
                        connectedF[E[s][i]] = 1;
                        return 1;
                }
        }

        return 0;
}

int matching() {
        int sol = 0;
        bio.resize(hor.size());
        connectedF.resize(vert.size());
        fill(connectedF.begin(), connectedF.end(), -1);

        for (int i = 0; i < hor.size(); ++i) {
                fill(bio.begin(), bio.end(), 0);
                sol += dfs(i);
        }

        return sol;
}

/************************
 *     Min cost max flow
 * *********************/
// from WA library
#include <iostream>
#include <algorithm>
#include <queue>

using namespace std;

const int N = 205, INF = 10000000;
// cost[i][j] == cost[j][i] always!
int graph[N][N], cost[N][N], reduced_cost[N][N];
int potential[N], prev[N], source = N - 1, sink = N - 2;

void reduce_cost() {
        for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                        if (graph[i][j] >= 0)
                                reduced_cost[i][j] += potential[i] - potential[j];
}

typedef pair<int, int> pii;

int dijkstra() {
        reduce_cost();

        fill(potential, potential + N, 2 * INF);
        fill(prev, prev + N, -1);
        priority_queue<pii, vector<pii>, greater<pii> > pq;

        pq.push(pii(0, source));
        potential[source] = 0;

        while (!pq.empty()) {
                pii v = pq.top(); pq.pop();
                int c = v.first, curr = v.second;

                if (potential[curr] < c) continue;

                for (int next = 0; next < N; next++) {
                        if (graph[curr][next] <= 0) continue;
                        if (potential[next] <= c + reduced_cost[curr][next]) continue;
                        potential[next] = c + reduced_cost[curr][next];
                        prev[next] = curr;
                        pq.push(pii(potential[next],next));
```

```
                }
        }
        return potential[sink];
}

int update(int& v) {
        int ret = INF;
        for (int c = sink, p = prev[c]; c != source; c = p, p = prev[c])
                ret = min(ret, graph[p][c]);
        for (int c = sink, p = prev[c]; c != source; c = p, p = prev[c])
                v += cost[p][c] * ret, graph[p][c] -= ret, graph[c][p] += ret;
        return ret;
}


int min_cost_max_flow(int& c) {
        int flow = 0; c = 0;

        fill(potential, potential + N, INF);
        copy(cost[0], cost[N], reduced_cost[0]);
        potential[source] = 0;
        for (int k = 0; k < N; k++)
                for (int i = 0; i < N; i++)
                        for (int j = 0; j < N; j++)
                                if (graph[i][j] > 0)
                                        potential[j] = min(potential[j], potential[i] + cost[i][j]);

        while (dijkstra() < INF) flow += update(c);
        return flow;
}


/**************************
 *     Network flow
 * ************************/
// tested on 11082 ACM problem
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <queue>
#define INF 0x3f3f3f

using namespace std;

int cap[42][42];
int how[42], ff[42];

int bfs(int source, int sink) {
        memset(how, -1, sizeof how);
        memset(ff, 0, sizeof ff);
        how[source] = source;
        ff[source] = INF;

        queue <int> Q;
        Q.push(source);

        while (Q.size()) {
                int s = Q.front(); Q.pop();

                if (s == sink) break;

                for (int i = 0; i < 42; ++i) {
                        if (cap[s][i] != 0 && how[i] == -1) {
                                ff[i] = min(ff[s], cap[s][i]);
                                how[i] = s;
                                Q.push(i);
                        }
                }
        }

        return ff[sink];
}

void flow(int source, int sink) {
        int maxflow = 0;
        for (int f = 0; f = bfs(source, sink); maxflow += f) {
                for (int s = sink; s != source; s = how[s]) {
                        cap[how[s]][s] -= f;
                        cap[s][how[s]] += f;
                }
```

```
        }
}

/**************************
 *     Number theory
 * ************************/
// WA library.
// CRT NOT TESTED
// extended gcd works
#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <algorithm>

using namespace std;

ldiv_t div_correct(long y, long x) {
        ldiv_t v = ldiv(y, x);
        if (y < 0 && v.rem != 0) {
                v.quot -= 1;
                v.rem += labs(x);
        }
        return v;
}

pair<long, long> extended_gcd(long a, long b) {
        if (a % b == 0)
                return pair<long, long>(0, 1);
        else {
                ldiv_t v = div_correct(a, b);
                pair<long, long> t = extended_gcd(b, v.rem);
                return pair<long,long>(t.second, t.first - t.second * v.quot);
        }
}

long crt(long *a, long *n, long r)
{
        long N = 1;
        for (int k = 0; k < r; k++)
                N *= n[k];

        long s = 0;
        for (int k = 0; k < r; k++)
        {
                long p = N / n[k];
                long x = extended_gcd(p, n[k]).first;
                s += a[k] * p * x;
                s %= N;
        }
        return s;
}

int main() {
        long A, B;
        while (scanf("%ld%ld", &A, &B) != EOF) {
                pair<long,long> xy = extended_gcd(A, B);
                printf("%ld %ld %ld\n", xy.first, xy.second, A * xy.first + B * xy.second);
        }
}
/**************************
 *     SCC
 * ************************/
// Tested on 11504
#include <cstdio>
#include <cstring>
#include <vector>
#define MAX 100000

using namespace std;

vector <vector <int> > E;
int on_stack[MAX], visited[MAX], component[MAX];
int num_components;
int global_time;
vector <int> node_stack;

void load() {
    int n, m;
    scanf("%d%d", &n, &m);
    E.clear();
```

```
    E.resize(n);

    for (int i = 0; i < m; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        E[a-1].push_back(b-1);
    }
}

int dfs(int s) {
    int lowlink = visited[s] = global_time++;
    node_stack.push_back(s);
    on_stack[s] = 1;

    for (int i = 0; i < E[s].size(); ++i) {
        if (!visited[E[s][i]]) {
            lowlink = min(lowlink, dfs(E[s][i]));
        } else if (on_stack[E[s][i]]) {
            lowlink = min(lowlink, visited[E[s][i]]);
        }
    }

    if (lowlink == visited[s]) {
        // s defines new component consisting of nodes on stack
        ++num_components;
        while (true) {
            int t = node_stack.back();
            component[node_stack.back()] = num_components;
            on_stack[node_stack.back()] = 0;
            node_stack.pop_back();
            if (t == s) break;
        }
    }

    return lowlink;
}

/**************************
 *     Tournament tree
 * ************************/
// NOT YET TESTED
// igor's new code
// supports:
// * find minimum in a range
// * change an element
#include <cstdio>
#include <algorithm>
#define MAXN 1000000
#define INF 0x3f3f3f3f

using namespace std;

int tournament[2*MAXN + 1];
int tt_size;
int A[MAXN];

void tt_create(int n) {
    for (tt_size = 1; tt_size < n; tt_size *= 2);

    for (int i = tt_size; i < tt_size*2; ++i) {
        if (i-tt_size < n) tournament[i] = A[i-tt_size];
        else tournament[i] = INF;
    }
    for (int i = tt_size - 1; i >= 1; --i) {
        tournament[i] = min(tournament[2*i], tournament[2*i+1]);
    }
}

int tt_change(int index, int new_value) {
    tournament[tt_size + index] = new_value;

    for (int i = tt_size + index; i >= 1; i /= 2) {
        tournament[i] = min(tournament[2*i], tournament[2*i+1]);
    }
}

// [from, to> [lo, hi>
int _tt_query(int from, int to, int p, int lo, int hi) {
    if (to <= lo || from >= hi) return INF;
    if (from <= lo && to >= hi) return tournament[p];
```

```
    return min(_tt_query(from, to, 2*p, lo, (lo+hi)/2), _tt_query(from, to, 2*p + 1, (lo+hi)/2, hi));
}

int tt_query(int from, int to) {
    return _tt_query(from, to, 1, 0, tt_size);
}
/*************************
 *     Union find
 * ************************/
// NOT TESTED YET
// C/P from Univ of Zagreb library
#include <cstdio>
#define MAXN 1000000
#define NOT_CONNECTED 0
#define CONNECTED 1
#define ALREADY_CONNECTED 2

int dad[MAXN], rank[MAXN];
// int kids[MAXN]; // if we want to find largest componenet

int union_find(int a, int b, bool connect = true) {
    int topa, topb;
    int newtop;

    for (topa = a; topa != dad[topa]; topa = dad[topa]);
    for (topb = b; topb != dad[topb]; topb = dad[topb]);
    dad[a] = topa; dad[b] = topb;

    if (topa != topb && connect) {
        if (rank[topa] > rank[topb]) {
            // kids[topa] += kids[topb];
            dad[topb] = newtop = topa;
        } else {
            // kids[topb] += kids[topa];
            dad[topa] = newtop = topb;
            if (rank[topa] == rank[topb]) rank[topb]++;
        }

        int x;
        for ( ; a != topa; ) x = dad[a], dad[a] = newtop, a = x;
        for ( ; b != topb; ) x = dad[b], dad[b] = newtop, b = x;

        return CONNECTED;
    } else {
        int x;
        for ( ; a != topa; ) x = dad[a], dad[a] = topa, a = x;
        for ( ; b != topb; ) x = dad[b], dad[b] = topb, b = x;

        return connect || topa == topb ? ALREADY_CONNECTED : NOT_CONNECTED;
    }
}

void union_find_init(int n) {
    for (int i = 0; i < n; ++i) {
        dad[i] = i;
        rank[i] = 0;
        // kids[i] = 1;
    }
}
```