```cpp
/*********************
 * 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;
```

```cpp
        }

        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;
}

int main() {
    int tilenum = 1;
    while (load()) {
        printf("Tile #%d\nWasted Space = %.2lf %%\n\n", tilenum++,  (1.0 - (P.area() / P
.convexHull().area())) * 100.0);
    }
    return 0;
```

```
}
```

```
#include <cmath>
#include <complex>
#include <vector>
#include <map>

using namespace std;

#define EPS 1e−9
#define INF 1000000000

typedef complex<double> pt;
typedef pair<pt, double> circle;
typedef pair<pt, pt> line;
typedef vector<pt> polygon;
typedef line seg;

#define det(a, b) imag(conj(a)*(b))
#define dot(a, b) real(conj(a)*(b))
#define sign(a) (abs(a) < EPS ? 0 : a > 0 ? 1 : −1)
#define signstar(a) (sign(a) == −1 ? −1 : 1)

pt xLineLine(line a, line b)
{
    return
        ( det(a.first, a.second) * (b.first − b.second) − det(b.first, b.second)
 * (a.first − a.second) ) / det(a.first − a.second, b.first − b.second) ;
}

bool xPtSeg(pt p, seg l)
{
    return abs(abs(p − l.first) + abs(p − l.second) − abs(l.first − l.second)) <
 EPS;
}

bool xPtSeg_open(pt p, seg l)
{
    return
        abs(p − l.first) > EPS &&
        abs(p − l.second) > EPS &&
        xPtSeg(p, l);
}

bool parallel(line a, line b)
{
    return abs(det(a.first − a.second, b.first − b.second)) < EPS;
}

bool xLineSeg(line a, seg b, pt &x)
{
    x = xLineLine(a, b);
    return !parallel(a, b) && xPtSeg(x, b);
}

bool xLineSeg_open(line a, seg b, pt &x)
{
    x = xLineLine(a, b);
    return !parallel(a, b) && xPtSeg_open(x, b);
}
```

```cpp
bool xPtLine(pt p, line l)
{
    double
        da = abs(p - l.first),
        db = abs(p - l.second),
        dc = abs(l.first - l.second);

    return abs(2 * (da + db + dc) - max(da, max(db, dc))) < EPS;
}

double ccw(pt a, pt b, pt c)
{
    return det(a - b, c - a);
}

bool comp_pt(const pt a, const pt b)
{
    if (abs(real(a - b)) < EPS)
        return imag(b - a) > EPS;
    return real(b - a) > EPS;
}

/*
 * Assume p[0] == p[-1]
 * Tested: UVA 11460
 */
vector<polygon> xLinePoly(line l, polygon p)
{
    vector<polygon> x;
    vector<vector<int> > s(2, vector<int>());
    vector<int> z;
    map<int, int> z_map, pos;
    pt u;
    double c;

    for (int i = 0; i < p.size(); i++)
    {
        if (i > 0 && xLineSeg_open(l, line(p[i - 1], p[i]), u))
            p.insert(p.begin() + i, u);

        c = ccw(l.first, l.second, p[i]);
        if (c > -EPS)
            s[0].push_back(i);
        if (c < EPS)
            s[1].push_back(i);

        if (abs(c) < EPS)
            if (z.size() == 0 || comp_pt(p[z.back()], p[i]))
                z.push_back(i);
            else
            {
                int lo = 0, hi = z.size(), mid;
                while (hi - lo > 0)
                    if (comp_pt(p[i], p[z[mid = (hi + lo - 1) / 2]]))
                        hi = mid;
                    else
                        lo = mid + 1;
                z.insert(z.begin() + lo, i);
            }
```

```cpp
    }

    for (int i = 0; i < z.size(); i++)
        z_map[z[i]] = i;

    for (int k = 0; k < s.size(); k++)
    {
        if (s[k].front() != s[k].back())
            s[k].push_back(s[k].front());

        for (int i = 1; i < s[k].size(); i++)
            if (z_map.count(s[k][i - 1]) > 0 && z_map.count(s[k][i]) > 0)
                for (int j = z_map[s[k][i - 1]] + sign(z_map[s[k][i]] - z_map[s[
k][i - 1]]); j != z_map[s[k][i]]; j += sign(z_map[s[k][i]] - z_map[s[k][i - 1]])
)
                    s[k].insert(s[k].begin() + i++, z[j]);

        pos.clear();
        for (int i = 0; i < s[k].size(); i++)
        {
            if (pos.count(s[k][i]) != 0)
            {
                x.push_back(polygon());
                for (int j = pos[s[k][i]]; j < i; j++)
                    x.back().push_back(p[s[k][j = pos[s[k][j]]]]);
            }
            pos[s[k][i]] = i;
        }
    }

    for (int i = x.size() - 1; i >= 0; i--)
        if (x[i].size() < 3)
            x.erase(x.begin() + i);
        else
            x[i].push_back(x[i].front());

    return x;
}

/* True if p is on segment a-b.
 *  - Assume a != b
 *  - True at endpoints      */
bool xPtSeg(pt p, pt a, pt b)
{
    return
        abs(det(p-a, b-a)) < EPS &&
        dot(p-a, b-a) > -EPS &&
        dot(p-b, a-b) > -EPS ;
}

/* True if segment a-b intersects segment c-d
 *  -- True at endpoints. */
bool xSegSeg(pt a, pt b, pt c, pt d)
{
    double
        ta = det(c-a, d-a),
        tb = det(d-b, c-b),
        tc = det(a-c, b-c),
        td = det(b-d, a-d) ;
    return
        xPtSeg(a, c, d) ||
        xPtSeg(b, d, c) ||
        xPtSeg(c, a, b) ||
        xPtSeg(d, b, a) ||
        sign(ta) && sign(ta) == sign(tb) &&
        sign(tc) && sign(tc) == sign(td) ;
}

/* True if segment a-b intersects segment c-d
 *  -- False at endpoints.
 *  -- False if segments are parallel. */
bool xSegSeg_open(pt a, pt b, pt c, pt d)
{
    double
        ta = det(c-a, d-a),
        tb = det(d-b, c-b),
        tc = det(a-c, b-c),
        td = det(b-d, a-d) ;
    return
        sign(ta) && sign(ta) == sign(tb) &&
        sign(tc) && sign(tc) == sign(td) ;
}

/* True if segment a-b intersects segment c-d
 *  -- Assumes that colinear and corner cases never occur. */
bool xSegSeg_simple(pt a, pt b, pt c, pt d)
{
    return
        det(c-a, d-a) > EPS == det(d-b, c-b) > EPS &&
        det(a-c, b-c) > EPS == det(b-d, a-d) > EPS ;
}

/* True if segment a-b intersects segment c-d
 *  -- Assumes that colinear and corner cases never occur.
 *  -- Cheesy method using xLineLine
 *     This also applies to closed corners, xSegLine, etc, but
 *          colinear cases need to be a special case           */
// NOT TESTED
bool xSegSeg_simple2(pt a, pt b, pt c, pt d)
{
    pt x =
        ( det(a, b) * (c - d) - det(c, d) * (a - b) )
            / det(a-b, c-d) ;
    double s = real((x-a)/(b-a)),
           t = real((x-c)/(d-c)) ;
    return
        EPS < s && s < 1-EPS &&
        EPS < t && t < 1-EPS   ;
}

/* True if segment a-b intersects line --c-d--
 *  -- Assumes that colinear and corner cases never occur. */
// Tested by ICPC 2005 Finals - GSM
bool xSegLine_simple(pt a, pt b, pt c, pt d)
{
    return
        det(c-a,d-a) > EPS == det(d-b,c-b) > EPS ;
}
```

```
/* Intersection of line a-b and line c-d
 *  -- Returns an "invalid" complex if a-b c-d parallel.
 */
pt xLineLine(pt a, pt b, pt c, pt d)
{
    //assert( abs(det(a-b, c-d)) > EPS );

    return
        ( det(a, b) * (c - d) - det(c, d) * (a - b) )
            / det(a-b, c-d) ;
}

void perp_bisector(pt a, pt b, pt &m, pt &d)
{
    m = (a + b) / pt(2.0, 0.0);
    d = (b - a) * pt(0.0, 1.0);
}

/* Intersection of a line and a circle
 *  -- Returns the number of points of intersection, 0, 1 or 2
 *  -- Populates points a and b with the points of intersection
 *  Tested: UVA 11037
 */
int xLineCircle(line x, circle y, pt &a, pt &b)
{
    double dpl = det(x.second - x.first, y.first - x.first) / abs(x.second - x.f
irst);
    pt m, d;
    perp_bisector(x.first, x.second, m, d);
    pt i = y.first - d * dpl / abs(d);

    if (abs(abs(dpl) - y.second) < EPS)
    {
        a = i;
        return 1;
    }
    else if (abs(dpl) < y.second - EPS)
    {
        double h = sqrt(y.second * y.second - dpl * dpl);
        a = i + h * (x.second - x.first) / abs(x.second - x.first);
        b = i - h * (x.second - x.first) / abs(x.second - x.first);
        return 2;
    }
    else
    {
        return 0;
    }
}

/* Intersection of two circles
 *  -- Returns the number of points of intersection, 0, 1 or 2 or INF
 *  -- Populates points m and n with the points of intersection
 *  Tested: UVA 11037
 */
int xCircleCircle(circle x, circle y, pt &m, pt &n)
{
    double d = abs(x.first - y.first);
    if (abs(x.second - y.second) < EPS && abs(x.first - y.first) < EPS)
    {
        return INF;
    }
    else if (abs(x.second + y.second - d) < EPS)
    {
        m = (x.first + y.first) / pt(2.0, 0.0);
        return 1;
    }
    else if (d < x.second + y.second - EPS && d > abs(x.second - y.second) + EPS
)
    {
        double a = (x.second * x.second - y.second * y.second + d * d) / (2 * d)
;
        double h = sqrt(x.second * x.second - a * a);
        pt p = x.first + a * (y.first - x.first) / d;
        m = pt(real(p) + h * (imag(y.first) - imag(x.first)) / d, imag(p) - h *
(real(y.first) - real(x.first)) / d);
        n = pt(real(p) - h * (imag(y.first) - imag(x.first)) / d, imag(p) + h *
(real(y.first) - real(x.first)) / d);
        return 2;
    }
    else
    {
        return 0;
    }
}

/*$*/
int main()
{
    return 0;
}
/*$*/
```

```
#include <vector>
#include <complex>
#include <cmath>

using namespace std;

#define EPS 1E-9
#define det(a, b) imag(conj(a)*(b))
#define dot(a, b) real(conj(a)*(b))
#define sign(a) (abs(a) < EPS ? 0 : a > 0 ? 1 : -1)
#define signstar(a) (sign(a) == -1 ? -1 : 1)

#define PI (2.0 * acos(0.0))
#define sq(x) ((x) * (x))
#define law_of_cosines(a, b, c) acos(min(max((sq(a) + sq(b) - sq(c)) / (2 * (a)
* (b)), -1.0), 1.0))

typedef complex<double> pt;
typedef pair<pt, pt> line;
typedef vector<pt> polygon;
typedef line seg;

/*
 * Assume V[0] == V[-1]
 */
double signed_area(polygon &V) {
    double A = 0.0;
    for (unsigned i = 1; i < V.size(); i++)
        A += det(V[i - 1], V[i]);
    return A / 2;
}

/*
 * Assume V[0] == V[-1]
 */
pt centroid(polygon &V) {
    pt c = pt(0.0, 0.0);
    for (unsigned i = 1; i < V.size(); i++)
        c += (V[i - 1] + V[i]) * pt(det(V[i - 1], V[i]), 0.0);
    return c / pt(6 * signed_area(V), 0.0);
}

#define det(a, b) imag(conj(a)*(b))

/* Returns 2 * (area of polygon V)
 *   - Assumes V[0] == V[-1]
 */
double area_polygon(polygon &V) {
    double A = 0.0;
    for (unsigned i = 1; i < V.size(); i++)
        A += det(V[i - 1], V[i]);
    return abs(A);
}

/*   - Assumes convex V in ccw order
 *   - Assumes V[0] == V[-1]
 */
bool inside_convex(pt p, polygon& V) {
    for (unsigned i = 1; i < V.size(); i++)
```

```
        if (det(V[i] − V[i − 1], p − V[i − 1]) < −EPS)
            return false;
    return true;
}

/*
 * Tests whether p is in simple polygon V
 *   − Assumes V[0] == V[−1]
 *   − Assumes p does not intersect V
 *   − Assumes segment p−q does not intersect corners
 *   − Assumes q is large enough
 */
bool inside_polygon(pt p, polygon& V) {
    pt q = polar(1e8, 1.2345);
    int s = 0;
    for (int i = 1; i < V.size(); i++)
        s += xSegSeg(p, q, V[i − 1], V[i]);
    return s % 2 == 1;
}

double pack_in_cirlce(vector<int> &side_lengths)
{
    double max_lo = 0.0, max_hi = 0.0, lo, hi, r, alpha;
    int max_i;
    bool outside = true;

    for (int i = 0; i < side_lengths.size(); i++)
    {
        if (0.5 * side_lengths[i] > max_lo)
        {
            lo = max_lo = 0.5 * side_lengths[i];
            max_i = i;
        }
        hi = max_hi += 2.0 * side_lengths[i];
    }

    while (abs(hi − lo) > EPS || (outside && lo < max_lo + EPS))
    {
        if (outside && abs(hi − lo) < EPS && lo < max_lo + EPS)
        {
            lo = max_lo;
            hi = max_hi;
            outside = false;
        }

        r = (hi + lo) / 2.0;
        alpha = 0.0;

        for (int i = 0; i < n; i++)
            alpha += (outside || i != max_i ? 1.0 : −1.0) * law_of_cosines(r, r,
 L[i]);

        if (outside ? alpha < 2 * PI : alpha > EPS)
            hi = r;
        else
            lo = r;
    }

    double area = 0.0;
```

```
    for (int i = 0; i < n; i++)
        area += (outside || i != max_i ? 1.0 : −1.0) * 0.5 * r * r * sin(law_of_
cosines(r, r, L[i]));
    return area;
}

/*$*/
int main() {
    return 0;
}
/*$*/
```

```
/*********************
 *     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]);
        }
    }
}

int main() {
    while (load()) {
        generate_costs();
        H.matching();
        int sol = 0;
        for (int i = 0; i < houses.size(); ++i) {
            for (int j = 0; j < men.size(); ++j) {
                if (H.ret[i][j]) sol += dist(houses[i], men[j]);
            }
        }
        printf("%d\n", sol);
```

```
    }
    return 0;
}
```

kmp.cpp

```
/***************************
 *     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];
        }
    }
}


int main() {
    return 0;
}
```

matching.cpp

```
/***************************
 *     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]) con
tinue;

                        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], potenti
al[i] + cost[i][j]);

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

/*$*/
int main() {
        int ncases;
        cin >> ncases;
        for (int caseno = 1; caseno <= ncases; caseno++) {
                if (caseno != 1) cout << endl;
                int num_buildings, num_shelters;

                vector<pair<int, int> > buildings, shelters;
                cin >> num_buildings >> num_shelters;
                fill(graph[0], graph[N], 0);
                fill(cost[0], cost[N], 0);

                for (int i = 0; i < num_buildings; i++) {
                        int x, y, cap;
                        cin >> x >> y >> cap;
                        graph[source][i] = cap;
                        buildings.push_back(pair<int, int>(x, y));
                }

                for (int i = 0; i < num_shelters; i++) {
                        int x, y, cap;
                        cin >> x >> y >> cap;
                        graph[i + num_buildings][sink] = cap;
                        shelters.push_back(pair<int, int>(x, y));
                }

                for (int i = 0; i < num_buildings; i++)
                        for (int j = 0; j < num_shelters; j++) {
                                cost[i][num_buildings + j] =
                                        1 + abs(buildings[i].first - shelters[j]
.first)

                                        + abs(buildings[i].second - shelters[j].
second);

                                cost[num_buildings + j][i] = -cost[i][num_buildi
ngs + j];

                                graph[i][num_buildings + j] = INF;
                        }
                int c = 0, plan_cost = 0;
                min_cost_max_flow(c);

                for (int i = 0; i < num_buildings; i++)
                        for (int j = 0; j < num_shelters; j++) {
                                int p;
                                cin >> p;
                                plan_cost += p * cost[i][num_buildings + j];
                        }

                if (c != plan_cost) {
                        cout << "SUBOPTIMAL" << endl;
                        for (int i = 0; i < num_buildings; i++) {
                                for (int j = 0; j < num_shelters; j++) {
                                        cout << (j ? " " : "") << graph[num_buil
dings + j][i];
                                }
                                cout << endl;
                        }
                } else cout << "OPTIMAL" << endl;
        }
        return 0;
}
/*$*/
```

```
/****************************
 *      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;
        }
    }
}

int main() {
    int T;
    scanf("%d", &T);

    for (int tt = 1; tt <= T; ++tt) {
        memset(cap, 0, sizeof cap);
        int R, C;
        scanf("%d%d", &R, &C);
        int before = 0;
        for (int i = 0; i < R; ++i) {
            int a; scanf("%d", &a);
            cap[0][i+1] = a - C - before;
            before = a;
        }
        before = 0;
        for (int i = 0; i < C; ++i) {
            int a; scanf("%d", &a);
            cap[R+i+1][R+C+1] = a - R - before;
            before = a;
        }
        for (int i = 0; i < R; ++i) {
            for (int j = 0; j < C; ++j) {
                cap[i+1][j+R+1] = 19;
            }
        }

        flow(0, R+C+1);

        printf("Matrix %d\n", tt);
        for (int i = 0; i < R; ++i) {
            for (int j = 0; j < C; ++j) {
                printf("%d ", cap[j+R+1][i+1] + 1);
            }
            printf("\n");
        }
        printf("\n");

    }
    return 0;
}
```

```
/****************************
 *      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;
}

int main() {
```

```
    int T;
    scanf("%d", &T);
    while (T--) {
        load();
        memset(visited, 0, sizeof visited);
        memset(on_stack, 0, sizeof on_stack);
        memset(component, 0, sizeof component);
        global_time = 1;
        num_components = 0;
        for (int i = 0; i < E.size(); ++i) {
            if (!visited[i]) {
                dfs(i);
            }
        }

        // solution is number of componenets with in-degree == 0
        vector <int> knock_down(num_components, 1);
        int sol = num_components;

        for (int i = 0; i < E.size(); ++i) {
            for (int j = 0; j < E[i].size(); ++j) {
                if (component[i] != component[E[i][j]] && knock_down[component[E
[i][j]]]) {
                    knock_down[component[E[i][j]]] = 0;
                    --sol;
                }
            }
        }

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

```cpp
/***************************
 *     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);
}
```

```cpp
/***************************
 *     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;
    }
}

int main() {
    return 0;
}
```