```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <utility>

using namespace std;

// DEFINICIONES DE TIPO
typedef pair<int, int> posicion;
typedef int altura;
typedef int poblacion;
typedef pair<altura, poblacion> zona;
typedef vector<vector<zona > > mapa;

vector<zona> generarVecinos(mapa m, posicion p) {
    vector<zona> res;

    if (p.first < m.size() - 1) {
        res.push_back(m[p.first + 1][p.second]);
    }

    if (p.first > 0) {
        res.push_back(m[p.first - 1][p.second]);
    }

    if (p.second < m[0].size() - 1) {
        res.push_back(m[p.first][p.second + 1]);
    }

    if (p.second > 0) {
        res.push_back(m[p.first][p.second - 1]);
    }
    return res;
}
bool esValle(mapa m, posicion p){
    vector<zona> zonasVecinas = generarVecinos(m, p);
    int cantVecinos = zonasVecinas.size();

    int i = 0;
    while (i < cantVecinos && zonasVecinas[i].first >= m[p.first][p.second].first) {
        i++;
    }
    return i == cantVecinos;
}

int main() {
    /* No hace falta modificar el main */

    // La posición
    int pos1;
    cin >> pos1;
    int pos2;
    cin >> pos2;
    posicion p = make_pair(pos1, pos2);

    // El mapa
    int f;
    cin >> f;
    int c;
    cin >> c;
    mapa m;
    char aux;
    int alt;
    int pob;
    for (int i = 0; i < f; i++) {
        vector<zona> fila;
        for (int j = 0; j < c; j++) {
            cin >> aux;
            cin >> alt;
            cin >> aux;
            cin >> pob;
```

```
            cin >> aux;
            fila.push_back(make_pair(alt, pob));
        }

        m.push_back(fila);
    }

    // Evaluo si la posición es valle
    bool res = esValle(m, p);

    // Imprimo la salida
    cout << (res?"True":"False") << endl;

    return 0;
}

```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <utility>

using namespace std;

// DEFINICIONES DE TIPO
typedef pair<int, int> posicion;
typedef int altura;
typedef int poblacion;
typedef pair<altura, poblacion> zona;
typedef vector<vector<zona > > mapa;

void miPropioSwap(vector<posicion> &v, int i, int j) {
    posicion tmp = v[i];
    v[i] = v[j];
    v[j] = tmp;
}

int poblacionDePos(mapa m, posicion p){
    return m[p.first][p.second].second;
}

void ordenar(vector<posicion> &v, mapa m2) {
    // Bubble Sort
    for (int i = 0; i < v.size(); i++) {
        for (int j = v.size() - 1; j > i; j--) {
            if (poblacionDePos(m2, v[j]) < poblacionDePos(m2, v[j - 1])) {
                miPropioSwap(v, j, j - 1);
            }
        }
    }
}


void reiniciarVector(vector<posicion> &v) {
    int i = 0;
    while (i < v.size()) {
        v.pop_back();
        i++;
    }
}

int crecimientoPoblacional(mapa m1, mapa m2, posicion p) {
    return m2[p.first][p.second].second - m1[p.first][p.second].second;
}

vector<posicion> bajaNatalidad(mapa m1, mapa m2) {
    // Tomo como mínimo el crecimiento poblacional de la primera pos
    int min = crecimientoPoblacional(m1, m2,make_pair(0,0));
    vector<posicion> res;

    for (int i = 0; i < m1.size(); i++) {
        for (int j = 0; j < m1[0].size(); j++) {
            posicion p = make_pair(i, j);
            if (crecimientoPoblacional(m1, m2, p) == min) {
                res.push_back(p);
            } else if (crecimientoPoblacional(m1, m2, p) < min) {
                reiniciarVector(res);
                min = crecimientoPoblacional(m1, m2, p);
                res.push_back(p);
            }
        }
    }

    ordenar(res, m2);

    return res;
}

```

```
73 int main() {
74     /* No hace falta modificar el main */
75 
76     // El mapa 1
77     int f;
78     cin >> f;
79     int c;
80     cin >> c;
81     mapa m;
82     char aux;
83     int alt;
84     int pob;
85     for (int i = 0; i < f; i++) {
86         vector<zona> fila;
87         for (int j = 0; j < c; j++) {
88             cin >> aux;
89             cin >> alt;
90             cin >> aux;
91             cin >> pob;
92             cin >> aux;
93             fila.push_back(make_pair(alt, pob));
94         }
95 
96         m.push_back(fila);
97     }
98 
99     // El mapa 2
100    int f2;
101    cin >> f2;
102    int c2;
103    cin >> c2;
104    mapa m2;
105    char aux2;
106    int alt2;
107    int pob2;
108    for (int i = 0; i < f2; i++) {
109        vector<zona> fila2;
110        for (int j = 0; j < c2; j++) {
111            cin >> aux2;
112            cin >> alt2;
113            cin >> aux2;
114            cin >> pob2;
115            cin >> aux2;
116            fila2.push_back(make_pair(alt2, pob2));
117        }
118 
119        m2.push_back(fila2);
120    }
121 
122    // Invoco la función
123    vector<posicion> pos = bajaNatalidad(m, m2);
124 
125    // Ordeno el vector
126   // Deja de ser necesario porque ahora va ordenado de acuerdo a especificación std::sort(pos
    .begin(), pos.end());
127 
128    // Imprimo la salida
129    for (int i = 0; i < pos.size(); i++) {
130        cout << "(" << pos[i].first << "," << pos[i].second << ") ";
131    }
132 
133    return 0;
134 }
135 
```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <utility>

using namespace std;

// DEFINICIONES DE TIPO
typedef pair<int, int> posicion;
typedef int altura;
typedef int poblacion;
typedef pair<altura, poblacion> zona;
typedef vector<vector<zona > > mapa;

void vivenEnAltura(mapa m, int &alt, int &cantGente) {
    // Tomo como altura máxima a la altura de la primera pos
    int max = m[0][0].first;
    int sumaPoblacion = 0;

    for (int i = 0; i < m.size(); i++) {
        for (int j = 0; j < m[0].size(); j++) {
            int alturaPos = m[i][j].first;
            int poblacionPos = m[i][j].second;
            if (alturaPos == max) {
                sumaPoblacion += poblacionPos;
            } else if (alturaPos > max) {
                max = alturaPos;
                sumaPoblacion = poblacionPos;
            }
        }
    }

    alt = max;
    cantGente = sumaPoblacion;
}

int main() {
    /* No hace falta modificar el main */

    // El mapa 1
    int f;
    cin >> f;
    int c;
    cin >> c;
    mapa m;
    char aux;
    int alt;
    int pob;
    for (int i = 0; i < f; i++) {
        vector<zona> fila;
        for (int j = 0; j < c; j++) {
            cin >> aux;
            cin >> alt;
            cin >> aux;
            cin >> pob;
            cin >> aux;
            fila.push_back(make_pair(alt, pob));
        }

        m.push_back(fila);
    }

    // Invoco la función
    int altura;
    int cantGente;
    vivenEnAltura(m, altura, cantGente);

    // Imprimo la salida
    cout << altura << " " << cantGente << endl;

    return 0;
```

```
73 }
74
```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <utility>

using namespace std;

// DEFINICIONES DE TIPO
typedef pair<int, int> posicion;
typedef int altura;
typedef int poblacion;
typedef pair<altura, poblacion> zona;
typedef vector<vector<zona > > mapa;

vector<zona> generarVecinos(mapa m, posicion p) {
    vector<zona> res;

    if (p.first < m.size() - 1) {
        res.push_back(m[p.first + 1][p.second]);
    }

    if (p.first > 0) {
        res.push_back(m[p.first - 1][p.second]);
    }

    if (p.second < m[0].size() - 1) {
        res.push_back(m[p.first][p.second + 1]);
    }

    if (p.second > 0) {
        res.push_back(m[p.first][p.second - 1]);
    }
    return res;
}
bool esValle(mapa m, posicion p){
    vector<zona> zonasVecinas = generarVecinos(m, p);
    int cantVecinos = zonasVecinas.size();
    int alturaPosActual = m[p.first][p.second].first;

    int i = 0;
    while (i < cantVecinos && zonasVecinas[i].first >= alturaPosActual) {
        i++;
    }
    return i == cantVecinos;
}

int promedioAlturasVecinas(mapa const &m, posicion const &p) {
    vector<zona> vecinos = generarVecinos(m, p);
    int cantVecinos = vecinos.size();
    int sumaAlturasVecinas = 0;
    for (int i = 0; i < cantVecinos; i++) {
        sumaAlturasVecinas += vecinos[i].first;
    }

    return sumaAlturasVecinas / cantVecinos;
}

int rellenarValles(mapa &m) {
    mapa m0 = m;
    int sumaDifAlturas = 0;

    for (int i = 0; i < m.size(); i++) {
        for (int j = 0; j < m[0].size(); j++) {
            posicion p = make_pair(i, j);
            if (esValle(m0, p)) {
                m[i][j].first = promedioAlturasVecinas(m0, p);
                sumaDifAlturas += m[i][j].first - m0[i][j].first;
            }
        }
    }

```

```
    return sumaDifAlturas;
}

int main() {
    /* No hace falta modificar el main */

    // El mapa 1
    int f;
    cin >> f;
    int c;
    cin >> c;
    mapa m;
    char aux;
    int alt;
    int pob;
    for (int i = 0; i < f; i++) {
        vector<zona> fila;
        for (int j = 0; j < c; j++) {
            cin >> aux;
            cin >> alt;
            cin >> aux;
            cin >> pob;
            cin >> aux;
            fila.push_back(make_pair(alt, pob));
        }

        m.push_back(fila);
    }

    // Invoco la función
    int res = rellenarValles(m);

    // Imprimo la salida
    cout << res << endl;
    cout << m.size() << endl;
    cout << m[0].size() << endl;
    for (int i = 0; i < m.size(); i++) {
        for (int j = 0; j < m[0].size();j++) {
            cout << "("<< m[i][j].first << "," << m[i][j].second << ") ";
        }
        cout << endl;
    }

    return 0;
}

```