WA.

3.5

```
//
// Daniel López Marqués
// A41
// A47
//

#include <iostream>
#include <fstream>
#include <vector>

// Explicación del árbol de exploración empleado y del vector solución
//El arbol de exporacion tiene numPuestos niveles, y por cada nivel numPersonas ramas
//El vector solucion es de tamaño numPuestos y por cada posicion se prueba numPersonas
índices


// Explicación de la estimación empleada
//Consiste en encontrar las satisfacciones máximas de cada puesto, y calcular un vector de
acumulados.
//Este vector se utiliza para estimar cual es la satisfaccion maxima
//que se puede conseguir desde la posicion actual hasta el final, y así comprobar si compensa
hacer la llamada recursiva



using tMatriz = std::vector<std::vector<int>>;

struct tDatos {
    int numPersonas;
    int numPuestos;
    int idenHada;
    tMatriz  satis;
};

struct tSol {
    std::vector<int> sol;
    std::vector<int> solMejor;
    int satisActual;
    int satisMejor;
    std::vector<int> marcas;
    int puestosMalos;
    int contadorMejor;
    bool hadaMala;
};

// Funciones implementadas por el alumno

bool esValida(int i, int k, tSol& s, tDatos& d) {
    bool satisfaccionPositiva = d.satis[k][i] >= 0;
```

cont k: cont & -0.2.

```
    bool marcado = s.marcas[i] > 1;
    return satisfaccionPositiva && !marcado;
}

bool esSolucion(tSol& s, tDatos& d) {
    bool hadaMala = s.hadaMala;
    bool muchosPuestosMalos = s.puestosMalos > (int)(d.numPuestos / 3);
    return hadaMala && !muchosPuestosMalos;
}

void resolver(int m, int n, int k, tSol& s, tDatos& d, std::vector<int>const& acum) {
    for (int i = 0; i < m; i++) {
        s.sol[k] = i;
        s.satisActual += d.satis[k][i];
        if (i == d.idenHada) s.hadaMala = true;
        if (i == k) s.puestosMalos++;
        s.marcas[i]++;
        if (esValida(i, k, s, d)) {
            if (k == n - 1) {
                if (esSolucion(s, d)) {
                    if (s.satisActual > s.satisMejor) {
                        s.satisMejor = s.satisActual;
                        s.solMejor = s.sol;
                        s.contadorMejor = 1;
                    }
                    else if (s.satisActual == s.satisMejor) {
                        s.contadorMejor++;
                    }
                }
            }
            else {
                if (s.satisActual + acum[k + 1] > s.satisMejor)
                    resolver(m, n, k + 1, s, d, acum);
            }
        }
        s.marcas[i]--;
        if (i == k) s.puestosMalos--;
        if (i == d.idenHada) s.hadaMala = false;
        s.satisActual -= d.satis[k][i];
    }
}

// resuelve un caso de prueba, leyendo de la entrada la
// configuración, y escribiendo la respuesta
bool resuelveCaso() {
    tDatos d;
    std::cin >> d.numPersonas;
    if (d.numPersonas == 0) return false;
    std::cin >> d.numPuestos >> d.idenHada;
    // Lectura de la satisfaccion de las personas
    d.satis.assign(d.numPuestos, std::vector<int>(d.numPersonas));
    for (int i = 0; i < d.numPuestos; ++i)
        for (int j = 0; j < d.numPersonas; ++j)
            std::cin >> d.satis[i][j];
```

```
    tSol s;
    s.marcas.resize(d.numPersonas);
    s.puestosMalos = 0;
    s.satisActual = 0;
    s.satisMejor = 0;
    s.sol.resize(d.numPuestos);
    s.solMejor.resize(d.numPuestos);
    s.contadorMejor = 0;
    s.hadaMala = false;
    //vector de maximos
    std::vector<int> acum;
    for (int i = 0; i < d.numPuestos; ++i) {
        int aux = 0;
        for (int j = 0; j < d.numPersonas; ++j)
        {
            if (d.satis[i][j] > aux) aux = d.satis[i][j];
        }
        acum.push_back(aux);
    }
    //vector de acumulados
    for (int i = acum.size() - 1; i > 0; i--) {
        acum[i - 1] += acum[i];
    }

    resolver(d.numPersonas, d.numPuestos, 0, s, d, acum);

    if (s.contadorMejor == 0) std::cout << "No" << "\n";
    else std::cout << s.satisMejor << " " << s.contadorMejor << "\n";
    // Dar valor a los parametros de la llamada a VA,
    // LLamada a la función de VA
    // Escribir los resultados




    return true;
}



int main() {

#ifndef DOMJUDGE
    std::ifstream in("datos3.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso())
        ;

#ifndef DOMJUDGE // para dejar todo como estaba al principio
    std::cin.rdbuf(cinbuf);
    system("PAUSE");
```

```
#endif

    return 0;
}
```