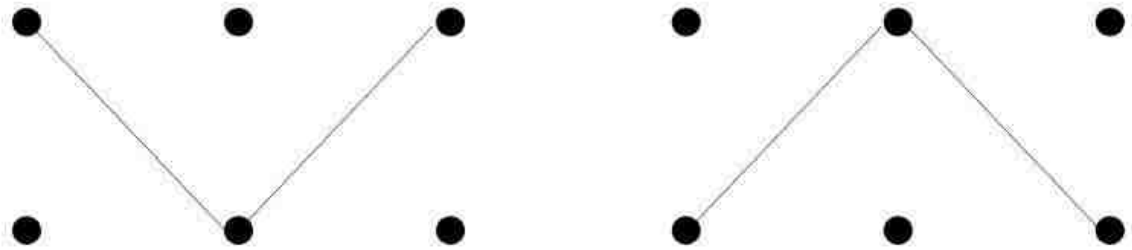
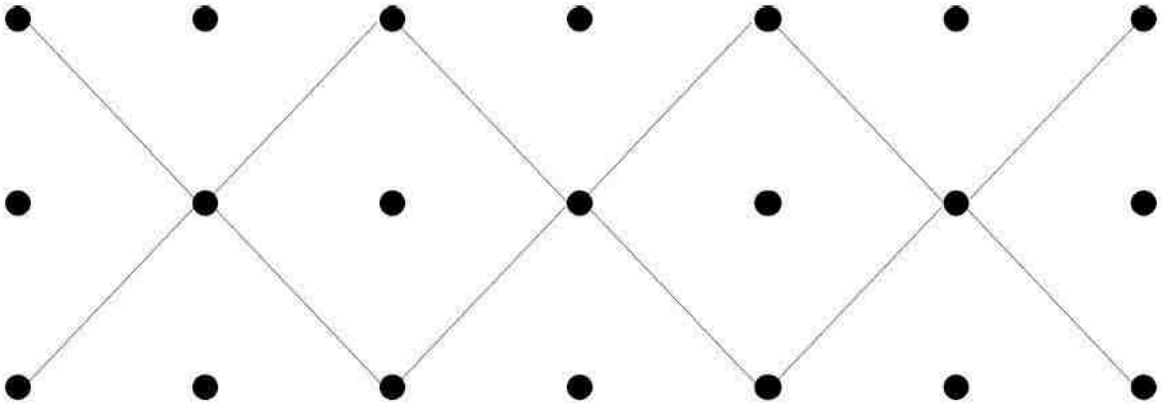
Diagonales

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité organizador)

Si dos cuadrados son adyacentes horizontalmente, para estar conectados, deben tener diagonales que vayan a la misma esquina.



Si extendemos esto a más cuadrados, podemos ver que una línea de cuadrados adyacentes ha de tener forma de secuencia de “montañas y valles”.



¿Y cómo podemos generalizar esto? La cuadrícula se puede pintar de dos maneras:

10101
01010
10101
...

o de la siguiente:

01010
10101
01010
...

(dejando las casillas con doses así, pero continuando con el patrón “ajedrez”). Estas son las únicas dos formas ya que si asignamos coordenadas *(x, y)* a los vértices, cada diagonal conecta dos vértices con la misma paridad de *x+y*, y por tanto no podemos tener vértices activos con paridades diferentes de *x+y*. La solución es probar ambas y ver si alguna conecta todos los vértices activos.

Y para comprobar si alguna los conecta, tenemos dos formas equivalentes. Tratamos los vértices como nodos de un grafo y las diagonales como sus aristas. El problema se convierte entonces en comprobar que todos los vértices activos forman parte de la misma componente conexa. Esto se puede hacer de dos formas equivalentes: a través de una DFS/BFS o usando UFDS. Este código usa la segunda pero ambas son igual de válidas.

Solución en C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int raiz(int a, vector<int> & par) {
	if (par[a] == a)
	return a;
	return par[a] = raiz(par[a], par);

}

void unir(int a, int b, vector<int> & par, vector<int> & rnk) {
	int raiza = raiz(a, par);
	int raizb = raiz(b, par);
	if (raiza == raizb)
	return;
	if (rnk[raiza] == rnk[raizb]) {
	par[raizb] = raiza;
	rnk[raiza]++;
	} else if (rnk[raiza] < rnk[raizb])
	par[raiza] = raizb;
	else
	par[raizb] = raiza;
}

bool conectado(vector<vector<int>> & grid) {
	int n = (int) grid.size(), m = (int) grid[0].size();
	vector<int> par((n+1)*(m+1));
	for (int i = 0; i <= n; i++)
	for (int j = 0; j <= m; j++)
		par[i*(m+1) + j] = i*(m+1) + j;
	vector<int> rnk((n+1)*(m+1), 0);
	int verticeActivo = -1;
	for (int i = 0; i < n; i++) {
	for (int j = 0; j < m; j++) {
		if (grid[i][j] == 0) {
		verticeActivo = i*(m+1) + j;
		unir(i*(m+1) + j, (i+1)*(m+1) + j + 1, par, rnk);
```

```
                } else if (grid[i][j] == 1) {
                verticeActivo = i*(m+1) + j+1;
                unir(i*(m+1) + j+1, (i+1)*(m+1) + j, par, rnk);
                }
        }
        }
        if (verticeActivo == -1)
        return true; // no hay vértices activos

        for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
                if ((grid[i][j] == 0 && raiz(i*(m+1) + j, par) != raiz(verticeActivo, par))
|| (grid[i][j] == 1 && raiz(i*(m+1) + j+1, par) != raiz(verticeActivo, par))) {
                return false;
                }
        }
        }
        return true;
}

void solve(vector<vector<int>> & grid) {
        int n = (int) grid.size(), m = (int) grid[0].size();
        vector<vector<int>> tableroActual(n, vector<int>(m));
        int dif1 = 0, dif2 = 0;
        for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
                if (grid[i][j] == 2)
                tableroActual[i][j] = 2;
                else {
                tableroActual[i][j] = (i+j)%2;
                dif1 += tableroActual[i][j] != grid[i][j];
                }
        }
        }
        bool posible1 = conectado(tableroActual);
        for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
                if (grid[i][j] == 2)
                tableroActual[i][j] = 2;
                else {
                tableroActual[i][j] = (i+j+1)%2;
                dif2 += tableroActual[i][j] != grid[i][j];
                }
        }
        }
        bool posible2 = conectado(tableroActual);
        if (posible1 && posible2)
        cout << min(dif1, dif2) << '\n';
        else if (posible1)
        cout << dif1 << '\n';
        else if (posible2)
        cout << dif2 << '\n';
        else cout << "-1\n";
}
```

```
int main()
{
	ios::sync_with_stdio(false);
	cin.tie(NULL);
	int t, n, m;
	cin >> t;
	while(t--) {
	cin >> n >> m;
	vector<vector<int>> grid(n, vector<int>(m));
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < m; j++) {
		cin >> grid[i][j];
		}
	}
	solve(grid);
	}
	return 0;
}
```

Autora de la solución: Blanca Huergo Muñoz (Miembro del comité organizador)