Cách giải

Liệt kê các miền liên thông gồm các ô cần tô cùng màu (gọi tắt là miền đơn sắc) (Đặt $M rồi cài DFS cho nhanh), đánh số các miền đơn sắc này từ 1 trở đi.

- Gọi $component[x, y]$ là số hiệu miền đơn sắc chứa ô (x, y)
- Gọi $color[i]$ là màu của miền đơn sắc thứ i, tức là màu của tất cả các ô (x, y) mà $component[x, y] = i$
- Gọi $area[i]$ là số ô (diện tích) của miền đơn sắc thứ i, tức là số ô (x, y) mà $component[x, y] = i$.

Xét tất cả các "khe" giữa hai ô kề cạnh, tức là với mỗi ô xét ô bên phải nó và bên dưới nó nếu có (tổng cộng $m(n-1) + n(m-1)$ khe như vậy). Với mỗi khe giữa hai ô khác màu ($(x, y)\&(x', y')$), tạo một bản ghi chứa 4 thông tin sau:

- Miền đơn sắc chứa ô (x, y): $cmp1 = component[x, y]$
- Màu của ô (x, y): $c_1 = a[x, y]$
- Miền đơn sắc chứa ô (x', y'): $cmp2 = component[x', y']$
- Màu của ô (x', y'): $c_2 = a[x', y']$

(Thực ra không cần lưu c_1 và c_2, chỉ cần $cmp1$ và $cmp2$ là đủ vì $c_1 = color[cmp1]$ và $c_2 = color[cmp2]$)

Lưu tất cả các bản ghi này lại, mỗi bản ghi ứng với một khe nối giữa hai miền đơn sắc $cmp1$ (màu c_1) và $cmp2$ (màu c_2). Không giảm tính tổng quát, giả sử $c_1 < c_2$ (Nếu lớn hơn thì ta đảo $cmp1$ và $cmp2$ cũng như đảo $c_1\&c_2$).

Kỹ thuật giảm độ phức tạp ở đây dựa vào nhận xét rằng số khe ≤ 2 triệu. Và để tìm miền liên thông lớn nhất gồm 2 màu XANH và ĐỎ thì ta chỉ cần xét các khe có c_1 =XANH và c_2 = ĐỎ mà thôi.

Sort lại danh sách các khe để các khe có cùng giá trị c_1 và c_2 dồn lại một đoạn liên tiếp trong dãy (đó là lý do ta quy ước $c_1 < c_2$). Xây dựng cấu trúc dữ liệu Disjoint-set forest (DSF) biểu diễn các tập miền đơn sắc, mỗi tập s trong CTDL này đi kèm với giá trị $weight[s]$ là tổng diện tích các miền đơn sắc ∈ tập.

- MakeSet(u): Tạo tập chỉ chứa 1 miền đơn sắc u:
  - $lab[u] \coloneqq 0$;
  - $weight[u] \coloneqq area[u]$;

- FindSet(u): Tìm tập chứa u, như bình thường (đi từ u lên gốc theo $lab[u]$, nén đường)
- Union(r, s): Hợp hai tập r, s. Chú ý khi hợp xong thì $weight$ của tập mới bằng tổng $weight$ hai tập cũ. Giá trị $weight[x]$ chỉ có nghĩa khi x là một gốc cây biểu diễn tập hợp trong DSF.
- **Khởi tạo:** Duyệt từng khe trong đoạn, với mỗi khe nối hai miền đơn sắc $cmp1 - cmp2$, gọi $MakeSet(cmp1)$ và $MakeSet(cmp2)$, cập nhật MaxArea bằng cái $weight$ lớn nhất vừa đặt.
- **Xử lý:** Duyệt từng khe trong đoạn, với mỗi khe $cmp1 - cmp2$ mà $FindSet(cmp1) = r \neq s = FindSet(cmp2)$ thì $Union(r, s)$. Cập nhật MaxArea theo $weight$ của tập hợp thành nếu nó lớn hơn MaxArea cũ.
- Kết thúc việc xử lý đoạn liên tiếp **MaxArea cho ta diện tích miền lớn nhất chỉ gồm hai màu c_1 và c_2.**

Còn lại thì xong rồi. Xử lý mỗi đoạn liên tiếp cho ta diện tích miền lớn nhất gồm 2 màu ứng với hai giá trị c_1, c_2 của các khe trong đoạn đó.

Độ phức tạp: Ngoài các thao tác tìm miền đơn sắc và sắp xếp khe. Việc xử lý một đoạn liên tiếp độ dài l mất thời gian $O(l \times \alpha(l))$. Tổng độ dài các đoạn liên tiếp ≤ số khe= $m(n-1) + n(m-1) \leq 2$ triệu).

Code tham khảo:

```cpp
#define taskname "COLORING"
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long lli;
const int maxMN = 1000;
const lli maxC = 1e6 + 1;

int m, n;
int a[maxMN][maxMN];
int lab[maxMN * maxMN], savelab[maxMN * maxMN];
int res;

inline int ReadInt()
{
    char c;
    for (c = getchar(); c < '0' || c > '9'; c = getchar());
    int res = c - '0';
    for (c = getchar(); c >= '0' && c <= '9'; c = getchar())
        res = res * 10 + c - '0';
    return res;
}

void WriteInt(int x)
{
    if (x > 9) WriteInt(x / 10);
    putchar(x % 10 + '0');
}

inline int Encode(int x, int y)
{
    return x * n + y;
}

inline void Decode(int code, int& x, int& y)
{
    x = code / n;
    y = code % n;
}

inline bool Valid(int x, int y)
{
    return 0 <= x && x < m && 0 <= y && y < n;
}

inline lli CPair(int c1, int c2)
{
    return c1 < c2 ? c1 * maxC + c2 : c2 * maxC + c1;
}

inline bool PureC(lli code)
{
    return code / maxC ==  code % maxC;
}

struct TEdge
{
    int u, v;
    lli code;
} e[maxMN * maxMN * 2];
```

int getchar(void)

```
int nEdges;
TEdge* pe[maxMN * maxMN * 2];

void Enter()
{
    m = ReadInt();
    n = ReadInt();
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
            a[i][j] = ReadInt();
}

void Init()
{
    nEdges = 0;
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
        {
            if (i > 0)
                e[nEdges++] = {Encode(i - 1, j),
                               Encode(i, j),
                               CPair(a[i - 1][j], a[i][j])
                              };
            if (j > 0)
                e[nEdges++] = {Encode(i, j - 1),
                               Encode(i, j),
                               CPair(a[i][j - 1], a[i][j])
                              };
        }
    ,
    for (int i = 0; i < nEdges; ++i)
        pe[i] = &e[i];
    sort(pe, pe + nEdges, [](const TEdge * x, const TEdge * y)
    {
        return x->code < y->code;
    });
    fill(begin(lab), end(lab), -1);
    res = 1;
}

int FindSet(int u)
{
    return lab[u] < 0 ? u : lab[u] = FindSet(lab[u]);
}

inline int Union(int r, int s)
{
    if (lab[s] < lab[r])
        swap(r, s);
    lab[r] += lab[s];
    lab[s] = r;
    return -lab[r];
}

inline void Update(int t)
{
    if (t > res) res = t;
}


void Solve1()
{
    TEdge* ptr = e;
    for (int i = 0; i < nEdges; ++i)
    {
        if (PureC(ptr->code))
        {
            int r = FindSet(ptr->u);
            int s = FindSet(ptr->v);
            if (r != s)
            {
                int Temp = Union(r, s);
                Update(Temp);
            }
        }
        ++ptr;
    }
    ptr = e;
    for (int i = 0; i < nEdges; ++i)
    {
        ptr->u = FindSet(ptr->u);
        ptr->v = FindSet(ptr->v);
        ++ptr;
    }
    copy(begin(lab), end(lab), begin(savelab));
}
```

```
void Solve2()
{
    int j = 0;
    lli OldCode = pe[0]->code;
    for (int i = 0; true; ++i)
        if (i == nEdges || pe[i]->code != OldCode)
        {
            //Union 2 areas
            for (int k = j; k < i; ++k)
            {
                int r = FindSet(pe[k]->u);
                int s = FindSet(pe[k]->v);
                if (r != s)
                {
                    int temp = Union(r, s);
                    Update(temp);
                }
            }
            //Restore labels
            for (int k = j; k < i; ++k)
            {
                lab[pe[k]->u] = savelab[pe[k]->u];
                lab[pe[k]->v] = savelab[pe[k]->v];
            }

            if (i == nEdges)
                break;
            OldCode = pe[i]->code;
            j = i;
        }
}
```

```
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    freopen(taskname".inp", "r", stdin);
    freopen(taskname".out", "w", stdout);
    Enter();
    Init();
    Solve1();
    Solve2();
    WriteInt(res);
}
```