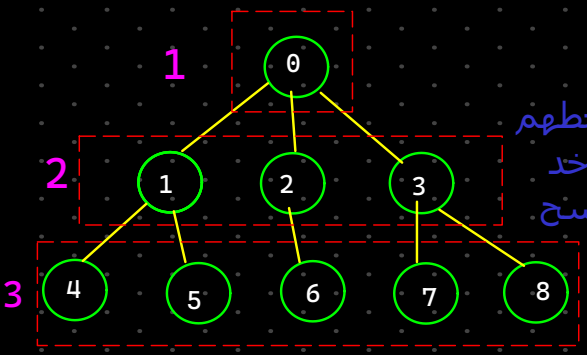
Graph Traversal
Breadth First Search
(BFS)

في ال DFS كنت باخد طريق واوصل بيه لحد الاخر لحد ما يخلص
وبعدين ارجع لكن هنا انا هاخد level by level



هبدأ بليفيل 1 هلاقي في نود واحد هاخد الneihbors بتوعها واحطهم
في queueوبعدين اعدي علي الكيو هلاقس فسه النودز 3 2 1 هاخد
النود1 واشوف اي جيرانها هلاقيهم 4 5 هضيفهم في اخر الكيو وامسح
ال 1واخش علي ال 2 وهكذا في الاخر الكيو هيبقي عامل كدا
4 5 6 7 8

من المسائل علي ال BFS اني اعرف اقصر طريق بين اي نود وكل النودز

```cpp
const int N = 1e5 +5, M=2e5+5, oo=0x3f3f3f3f;
int n , m ,u,v ;
vector<int> adj[N];

int dis[N];

void BFS(int source) {
    memset(dis , Val: oo,  Size: sizeof dis);
    dis[source] = 0 ;
    queue<int> q;
    q.push(source);

    while (! q.empty()) {
        int u = q.front();
        q.pop();

        for ( int v : adj[u]) {
            if(dis[v] == oo) {
                q.push(v);
                dis[v] = dis[u]+1 ;
            }
        }
    }
}
```

ال dis array هي اللي بيتخزن فيها
اقصر مسافه بين كل نود والsource

لازم اخزن الجراف في adj list عشان
اعرف اوصل لل neighbors بسرعه

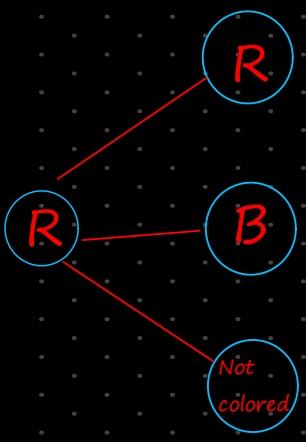
Time complexity

O(n+m)

اهم ميزه هنا اني بعد عن ال recursion

طيب لو انا عايز اعرف عدد ال connected components هو نفس كود ال DFS بس في تعديل انت كل مره هتروح تنادي علي ال BFS هيبقي فيها السطر بتاع ال memset اللي بيحط value لل array كلها فلازم تحط السطر دا فال main وتشيله من ال BFS وتعمل نفس كود ال dfs

في بروبلم حلوه انك معاك مجموعه من البيوت وعايزين نلونهم بلونين فقط بشرط مفيش بيتين جنب بعض نفس اللون و انت تقوله هل هينفع ولا مستحيل



وانا واقف عند اي نود هيبقي قدامي 3 احتمالات لل neighbors

لو انا متلون وجاري متلون نفس اللون هرجع false

لو انا متلون وجاري متلون لون تاني skip

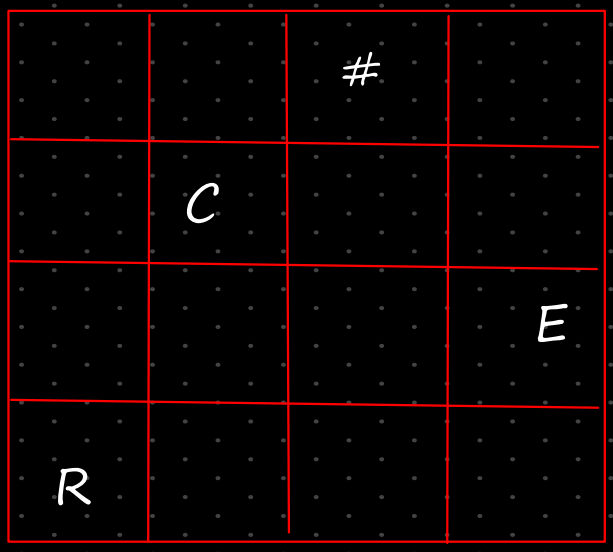
لو انا متلون وغيري مش متلون هلونه لون تاني غيري

```cpp
const int N = 1e5 +5, M=2e5+5, NOT_COLORED = 0, RED=1 , BLUE= 2;

int n , m ,u,v ;
vector<int> adj[N];

int color[N];

bool is_bi_colorable(int source) {
    memset(color , Val: NOT_COLORED, Size: sizeof color);
    color[source] = RED ;
    queue<int> q;
    q.push(source);

    while (! q.empty()) {
        int u = q.front();
        q.pop();

        for ( int v : adj[u]) {
            if(color[v] == NOT_COLORED) {
                color[v] = (color[u] == RED ? BLUE : RED) ;
                q.push(v);
            }
            else if(color[v] == color[u])  return false;
        }
    }
    return true;
}
```

في حته حلوه في المسئله دي بيقولك هي cycle its size is odd تعرفها بعينك لو في يبقي مينفعش

فاكر الجزء بتاع ال biportiite graph اللي قولنا انه جراف اقدر اقسمه لجزين مفيش بين نودز كل set اي edge هنا هقول ال 2sets هما الاحمر والازرق لو في اي edge بين الاحمر وبعضه يبقي false هي هي نفس المسأله دي

في مسأله مهمه جدا وهتفتح افكار كتير عندي *grid* وهيقولي كل خانه فيها قطه او فار او مصيده او فاضيه او باب خروج عندي كذا قطه لكن فار واحد وعايز اعرف هل الفار يستطيع الخروج ولا الفاره يمسكه الاول وهرجع



والاتنين بيتحركوا حركه واحده *true or false*
كل ثانيه اوممكن ميتحركش فانا عايز اشوف مين

هيوصل للـ*exit* الاول بحيث ان الفار ميبقاش عدي علي اي خانه القط وصل فيها قبله عشان ميبقاش مستنيه هناك وياكله

المسأله دي تعتبر *implicit* لان انا مش مديك نودز و*edges*جاهزين انت هت *generate*

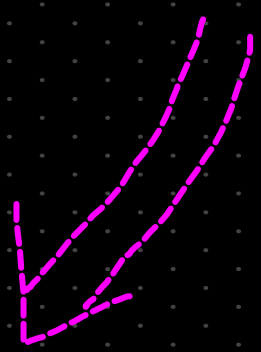
sol: اول حاجه هروح اخد الـ*grid* وعايز وان باخده اعرف فين مكان الفأر هنا انا حددت مكان الفار باني عرفت ال *row &col* اللي هو موجود فيهم

```
const int N = 1005;
char grid[N][N];
```

ثانيا انا هروح اعمل *grid* تانيه زي الاولي بس هسجل فيها القطط وصلت لكل *cell* في الثانيه رقم كام

```
int main() {
    int mR, mC;
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        cin >> grid[i];
        for (int j = 0; j < m; ++j) {
            if (grid[i][j] == 'M') {
                mR = i;
                mC = j;
            }
        }
    }
    BFSCats();
    cout << (BFSMouse(mR, mC) ? "YES" : "NO") << endl;
}
```

هسجل عند كل قطه رقم 0 لان هي وصلتلها في الثانيه 0 وبعدين اشوف الـ*cells* اللي جيران ليها واسجلهم في الثانيه 2 وهكذا وهستخدم ال *queue*عشان اعرف النود اللي شغال عليها بالظبط نفس كود ال *BFS*



```cpp
const int oo = 0x3f3f3f3f;
int n, m;

int catDis[N][N];
int mouseDis[N][N];

int dR[] = {1, -1, 0, 0};   // Steps to get neighbors
int dC[] = {0, 0, 1, -1};   // Steps to get neighbors

void BFSCats() {
    memset(catDis, Val: oo, Size: sizeof(catDis));

    queue<pair<int, int>> q;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (grid[i][j] == 'C') {
                q.push( x: {x: [&] i, y: [&] j});
                catDis[i][j] = 0;
            }
        }
    }

    while (!q.empty()) {
        pair<int, int> u = q.front();
        q.pop();

        for (int k = 0; k < 4; ++k) {
            int nR = u.first + dR[k];
            int nC = u.second + dC[k];
            if (nR < n && nR >= 0 && nC < m && nC >= 0 && catDis[nR][nC] == oo && grid[nR][nC] != '#') {
                catDis[nR][nC] = catDis[u.first][u.second] + 1;
                q.push( x: {x: [&] nR, y: [&] nC});
            }
        }
    }
}
```

وهروح اعمل نفس اللي انا لسا عامله

مع الcat تاني مع ال mouse بس تعديل بسيط اني هزود شرط اني مش هدخل

الcell دي غير لما اكون زمن وصولي يكون قبل القطه عشان متبقتش مستنياني

هناك ولو لقيت ال Exit هرجع true بس انا هكون باعتله مكان الفار

```cpp
bool BFSMouse(int mR, int mC) {
    memset(mouseDis, Val: oo, Size: sizeof(mouseDis));

    queue<pair<int, int>> q;
    q.push( x: {x: [&] mR, y: [&] mC});
    mouseDis[mR][mC] = 0;

    while (!q.empty()) {
        pair<int, int> u = q.front();
        q.pop();

        for (int k = 0; k < 4; ++k) {
            int nR = u.first + dR[k];
            int nC = u.second + dC[k];
            if (nR < n && nR >= 0 && nC < m && nC >= 0 && mouseDis[nR][nC] == oo && grid[nR][nC] != '#' && catDis[nR][nC] > mouseDis[u.first][u.second] + 1) {
                if (grid[nR][nC] == 'E') return true;

                mouseDis[nR][nC] = mouseDis[u.first][u.second] + 1;
                q.push( x: {x: [&] nR, y: [&] nC});
            }
        }
    }
    return false;
}
```