

**HỘI THẢO KHOA HỌC LẦN THỨ XIV CÁC TRƯỜNG CHUYÊN
KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ - NĂM 2021**

**CHUYÊN ĐỀ
BÀI TOÁN TÌM ĐƯỜNG ĐI NGĂN NHẤT
TRÊN ĐỒ THỊ LƯỚI**

Tháng 9 năm 2021

Mục lục

Mục lục	3
Mở đầu	6
1. Lý thuyết	6
1.1. Danh sách kè đồi với đồ thị lưới	6
1.2. Thuật toán BFS	7
1.2.1 Giới thiệu thuật toán BFS	7
1.2.2 Cấu trúc dữ liệu và giải thuật BFS	7
1.2.3 Thuật toán BFS trên đồ thị lưới (loang dầu)	8
1.3. Thuật toán Dijkstra	11
1.3.1. Giới thiệu thuật toán Dijkstra.....	11
1.3.2. Cấu trúc dữ liệu và giải thuật Dijkstra.....	11
1.3.3. Thuật toán Dijkstra trên đồ thị lưới	12
1.4. Thuật toán BFS 0-1	15
1.4.1. Giới thiệu thuật toán BFS 0-1	15
1.4.2. Cấu trúc dữ liệu và giải thuật BFS 0-1.....	15
1.4.3. Thuật toán BFS 0-1 trên đồ thị lưới.....	16
1.5. Dial's Algorthm.....	19
1.5.1. Giới thiệu thuật toán Dial's.....	19
1.5.2. Cấu trúc dữ liệu và giải thuật Dial's.....	19
1.5.3. Thuật toán Dial's trên đồ thị lưới	20
2. Ứng dụng.....	22
2.1. Ghép chữ (Đề thi DHĐBBB khối 11 năm 2021).....	22
2.1.1. Đề bài.....	22
2.1.2. Giải thuật.....	24
2.1.2. Cài đặt.....	25
2.2. Hồ thiên nga (COI 2005)	26
2.2.1. Đề bài.....	26
2.2.2. Giải thuật.....	27
2.2.3. Cài đặt.....	28
2.3. Covid'19 (Đề thi DHĐBBB 2020)	30
2.3.1. Đề bài.....	30
2.3.2. Giải thuật.....	31

2.3.3. Cài đặt.....	31
2.4. Sabor	33
2.4.1. Đề bài.....	33
2.4.2. Giải thuật.....	34
2.4.3. Cài đặt.....	34
2.5. Robot	36
2.5.1. Đề bài (Đề bài của thầy Nguyễn Đức Nghĩa)	36
2.5.1. Giải thuật.....	37
2.5.3. Cài đặt.....	38
2.6. Thuật toán Dial trên lưới.....	41
2.6.1. Đề bài.....	41
2.6.2. Giải thuật.....	42
2.6.3. Cài đặt.....	43
2.7. Super Computer	43
2.7.1. Đề bài (Đề bài của thầy Đỗ Đức Đông)	44
2.7.2. Giải thuật.....	45
2.7.3. Cài đặt.....	46
2.8. Camelot	47
2.8.1. Đề bài (Đề bài của thầy Đỗ Đức Đông)	47
2.8.2. Giải thuật.....	48
2.8.3. Cài đặt.....	48
2.9. Battle of Hogwarts	50
2.9.1. Đề bài.....	50
2.9.2. Giải thuật.....	51
2.9.3. Cài đặt.....	52
3. Bài tập tự luyện.....	54
3.1. Bài tập có hướng dẫn.....	54
3.1.1. Fire Again (CF35C)	54
3.1.2. You and Me (PUCMM223)	55
3.1.3. Meeting For Part (DCEPC706).....	55
3.1.4. Famous Grid (SPIRALGR)	55
3.1.4. Fillomino 2 (CF 1517C)	56
3.2. Bài tập khác	56
3.2.1. Minimum Knight moves (NAKANJ)	56

3.2.2. Maze Escape (11931)	56
3.2.3. SC (SC)	56
3.2.4. Đường đi (OLP_CT20_ROUTE)	56
3.2.5. Shortest distance between two cells in a matrix or grid.....	56
3.2.6. Shortest Path in a Binary Weight Graph.....	57
Kết luận	57
Tài liệu tham khảo.....	57

Mở đầu

Dạng bài về “Tìm đường đi ngắn nhất trên đồ thị lưới” cũng khá phổ biến, đây đều là các dạng bài không dễ, đòi hỏi học sinh có tư duy khá, nhiều bài đòi hỏi học sinh sáng tạo mới vận dụng được.

Học sinh cần có một số kiến thức để đảm bảo học được chuyên đề này là Đồ thị cơ bản, duyệt đồ thị; cấu trúc dữ liệu.

Nhận thấy đây là một chuyên đề thú vị và rất cần cho những học sinh tham gia các kì thi học sinh giỏi. Vì vậy, tôi quyết định tìm hiểu và viết chuyên đề: “Tìm đường đi ngắn nhất trên đồ thị lưới”.

Nhân đây, tôi cũng xin gửi lời cảm ơn tới các thầy cô, các bạn đồng nghiệp đã giúp đỡ, cung cấp tài liệu và góp ý để tôi bổ sung, hoàn thiện chuyên đề. Hi vọng rằng, chuyên đề sẽ cung cấp cho các bạn đồng nghiệp và các em học sinh một phần kiến thức bổ ích

Tất cả bộ Testcase và code mẫu của tất cả trong chuyên đề nằm trong link sau:

https://drive.google.com/drive/folders/10iruYRHlpG5Lwm6QdLtOLygMT_EwRr6p?usp=sharing

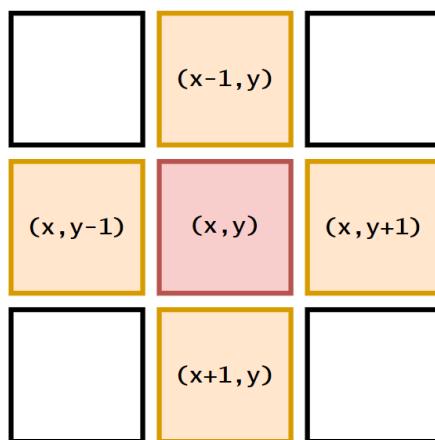
Quý thầy cô cũng có thể cho học sinh làm bài và chấm bài trực tiếp trên hệ thống lqdoj.edu.vn trong contest Grid Graph gồm các bài tập liên quan theo link sau: <https://lqdoj.edu.vn/contest/gridgraph>

1. Lý thuyết

1.1. Danh sách kè đối với đồ thị lưới

Đối với các bài toán, mà mỗi ô trên lưới **có thể đi đến** các ô **có cạnh kè** với nó.

Cụ thể từ ô có vị trí ở hàng x cột y gọi là (x, y) có thể đi đến $(x - 1, y)$, $(x, y - 1)$, $(x + 1, y)$ và $(x, y + 1)$. (với điều kiện nằm trong lưới)



Ta có thể cài đặt các định nghĩa sau, để thuận tiện cho việc xét các ô kè của (x, y)

- Hai mảng dx và dy

```
int dx[] = { 0, 0, -1, 1 };
int dy[] = { -1, 1, 0, 0 };
```

- Hàm $inGrid(x, y)$ – trả về **true** nếu ô (x, y) nằm trong lưới $n * m$, ngược lại trả về **false**

```
bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}
```

Áp dụng các định nghĩa trên để liệt kê các ô kề với (x, y)

```
cout << "Các ô kề với (x, y)\n";
for (int i = 0; i < 4; i++) {
    int u = x + dx[i], v = y + dy[i];
    if (!inGrid(u, v)) continue;
    cout << u << " " << v << "\n";
}
```

1.2. Thuật toán BFS

1.2.1 Giới thiệu thuật toán BFS

Thuật toán tìm kiếm theo chiều rộng (BFS) là một trong những thuật toán tìm kiếm cơ bản và quan trọng trên đồ thị.

Kết quả của thuật toán là danh sách các đường đi ngắn nhất từ đỉnh s bắt kè đến các đỉnh còn lại trong đồ thị (không trọng số).

Thuật toán BFS có độ phức tạp là $O(N + M)$, trong đó N là số đỉnh, M là số cạnh của đồ thị.

1.2.2 Cấu trúc dữ liệu và giải thuật BFS

- **Cấu trúc dữ liệu:**

Mảng d : Lưu lại giá trị của đường đi ngắn nhất ($d[i]$: giá trị đường đi ngắn nhất $s \rightarrow i$), ban đầu gán bằng ∞ .

Mảng $color$: đánh dấu các đỉnh trong đồ thị, ban đầu tất cả bằng **false**.

Hàng đợi q : hàng đợi các đỉnh trong đồ thị.

- **Thuật toán:**

Bước 1: Đưa đỉnh s vào hàng đợi q , gán $d[s] = 0$.

Bước 2: Lấy ra đỉnh u từ hàng đợi q .

Bước 3: Nếu đỉnh u đã được đánh dấu ($\text{color}[u] = \text{true}$) thì quay lại bước 2.

Bước 4: Đánh dấu đỉnh u ($\text{color}[u] \rightarrow \text{true}$). Duyệt các đỉnh v kề với đỉnh u . Nếu đỉnh v chưa được đánh dấu ($\text{color}[v] = \text{false}$) thì thêm đỉnh v vào trong hàng đợi và gán $d[v] = \min(d[v], d[u] + 1)$.

Bước 5: Nếu hàng đợi rỗng thì kết thúc thuật toán, ngược lại quay lại bước 2.

1.2.3 Thuật toán BFS trên đồ thị lưới (loang dầu)

- **Bài toán BFS on Grid:**

Cho một ma trận lưới $n * m$, có các ô bị chặn. Từ một ô vuông có thể đi sang 4 ô **kề cạnh** nhưng **không được đi vào** ô bị chặn. Viết chương trình tìm độ dài đường đi ngắn nhất từ (x, y) đến (u, v) .

- (i, j) là ô hàng i cột j .
- Ký tự $*$ là ô bị chặn.
- Ký tự $.$ là ô không bị chặn.
- Ký tự G là ô (x, y) .
- Ký tự R là ô (u, v) .
- Luôn tồn tại đường đi từ (x, y) đến (u, v) .

Dữ liệu

- Dòng đầu tiên: Hai số nguyên dương n và m
- n dòng tiếp theo: Mô tả ma trận lưới.
- Giới hạn: $n, m \leq 1000$

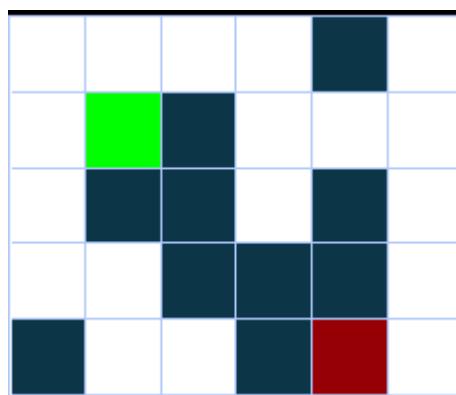
Kết quả

- Kết quả của chương trình.

Ví dụ:

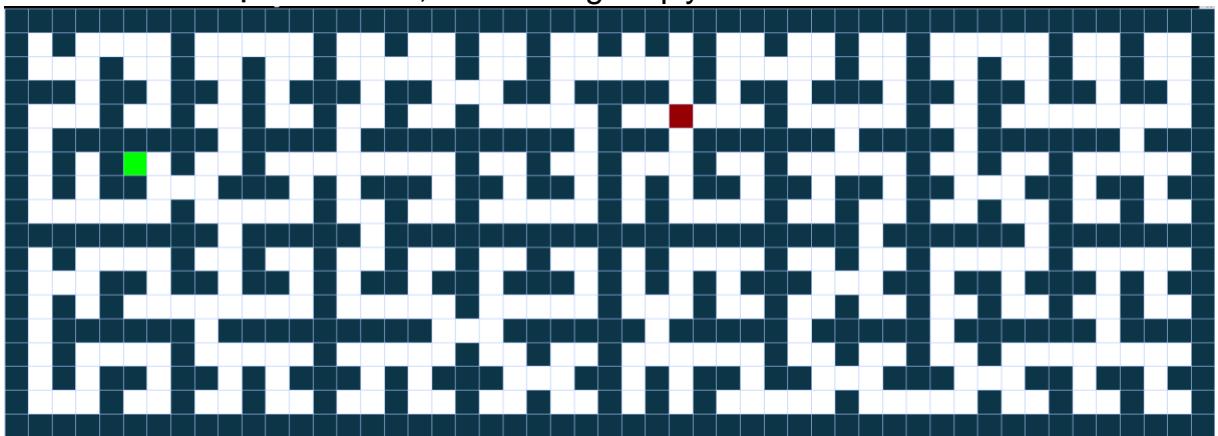
Input	Output
5 6*. .G***.*. .***. * .. *R.	10

Giải thích



• **Giải pháp:**

Xem các ô trong lưới như một đỉnh trong đồ thị. Hai đỉnh có cạnh với nhau nếu chúng kề cạnh và không phải ô bị chặn. Dựa vào cách cài đặt danh sách kề trên lưới và thuật toán BFS, ta có thể giải quyết bài toán.



(x, y) : MÀU XANH – (u, v) : MÀU ĐỎ

Tham khảo cách cài đặt sau:

```
#include <bits/stdc++.h>
using namespace std;
```

```

const int N = 1e3 + 5;
int dx[] = { -1, 1, 0, 0 };
int dy[] = { 0, 0, 1, -1 };
int n, m, X, Y, U, V;
int d[N][N]; // d[i][j] = độ dài đường đi từ ngắn nhất (x, y) -> (u, v)
char ch[N][N]; // lưới
queue<pair<int, int>> q; // hàng đợi - sử dụng CDTL pair (first - hàng, second - cột)
bool color[N][N]; // đánh dấu các ô trong lưới, ban đầu khởi tạo false
bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> ch[i][j];
            if (ch[i][j] == 'G')
                X = i, Y = j;
            else if (ch[i][j] == 'R')
                U = i, V = j;
            d[i][j] = n * m;
        }
    }

    q.push({ X, Y }); // đưa ô (X, Y) vào hàng đợi
    d[X][Y] = 0;
    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop(); // lấy ô (x, y) ra khỏi hàng đợi
        if (color[x][y]) continue;
}

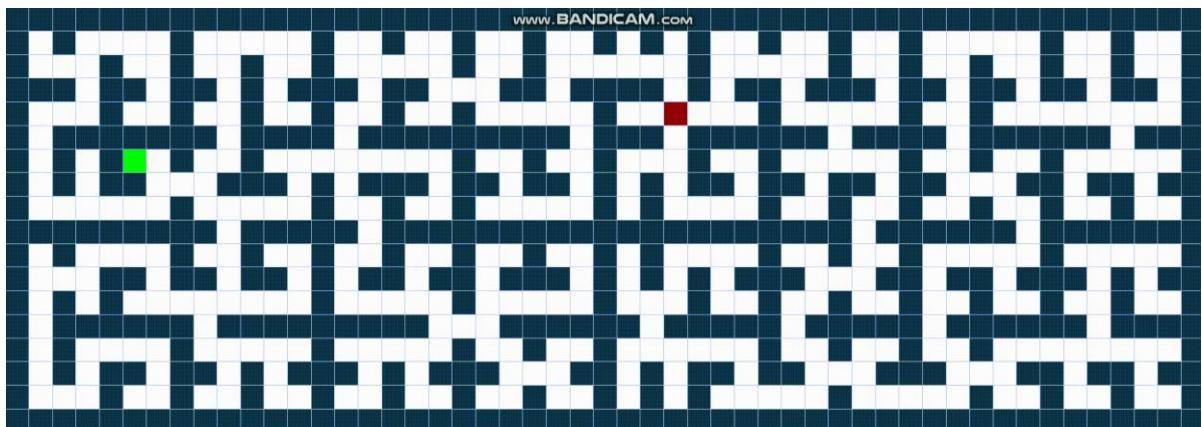
```

```

color[x][y] = true;
for (int i = 0; i < 4; i++) {
    int u = x + dx[i], v = y + dy[i];
    // xét ô (u, v) kề với (x, y)
    if (!inGrid(u, v) || color[u][v] || ch[u][v] == '*')
        continue;
    q.push({u, v});
    d[u][v] = min(d[u][v], d[x][y] + 1);
}
cout << d[U][V];
}

```

- *Mô phỏng thuật toán loang dầu:*



Có thể xem mô phỏng online: [BFS on Grid - Google Drive](#) (File dạng PDF không mô phỏng được)

1.3. Thuật toán Dijkstra

1.3.1. Giới thiệu thuật toán Dijkstra

Thuật toán Dijkstra là thuật toán tìm các đường đi ngắn nhất từ đỉnh s bắt kè đến các đỉnh còn lại trong đồ thị (có trọng số không âm).

Thuật toán Dijkstra có độ phức tạp là $O(M \log N)$, trong đó N là số đỉnh, M là số cạnh của đồ thị.

1.3.2. Cấu trúc dữ liệu và giải thuật Dijkstra

- **Cấu trúc dữ liệu:**

Mảng d : lưu lại giá trị của đường đi ngắn nhất ($d[i]$: giá trị đường đi ngắn nhất $s \rightarrow i$), ban đầu gán bằng ∞ .

Mảng $color$: đánh dấu các đỉnh trong đồ thị, ban đầu tất cả bằng *false*.

Hàng đợi có độ ưu tiên q : hàng đợi các đỉnh trong đồ thị, ưu tiên đỉnh i hơn đỉnh j nếu $d[i] < d[j]$.

- **Thuật toán:**

Bước 1: Đưa đỉnh s vào hàng đợi q , gán $d[s] = 0$.

Bước 2: Lấy ra đỉnh u từ hàng đợi q .

Bước 3: Nếu đỉnh u đã được đánh dấu ($\text{color}[u] = \text{true}$) thì quay lại bước 2.

Bước 4: Đánh dấu đỉnh u ($\text{color}[u] \rightarrow \text{true}$). Duyệt các đỉnh v kề với đỉnh u . Nếu đỉnh v chưa được đánh dấu ($\text{color}[v] = \text{false}$) thì thêm đỉnh v vào trong hàng đợi và gán $d[v] = \min(d[v], d[u] + \text{dist}(u, v))$.

Bước 5: Nếu hàng đợi rỗng thì kết thúc thuật toán, ngược lại quay lại bước 2.

1.3.3. Thuật toán Dijkstra trên đồ thị lưới

- **Bài toán Dijkstra on Grid:**

Cho một ma trận lưới $n * m$, các ô vuông được thể hiện bởi chữ số. Từ một ô vuông có thể đi sang 4 ô **kề cạnh**. Viết chương trình tìm đường đi từ (x, y) đến (u, v) có tổng các ô chữ số là nhỏ nhất.

- (i, j) là ô hàng i cột j .
- Ký tự G là ô (x, y) .
- Ký tự R là ô (u, v) .

Dữ liệu

- Dòng đầu tiên: Hai số nguyên dương n và m
- n dòng tiếp theo: Mô tả ma trận lưới.
- Giới hạn: $n, m \leq 1000$

Kết quả: Độ dài đường đi.

Ví dụ:

Input	Output
3 3 323 G9R	8

- ***Giải pháp:***

Xem các ô trong lưới như một đỉnh trong đồ thị có hướng. Hai đỉnh có cạnh với nhau nếu chúng kề cạnh và có trọng số là chỉ số làm chậm của ô kết thúc. Dựa vào cách cài đặt danh sách kề trên lưới và thuật toán Dijkstra, ta có thể giải quyết bài toán.



(x, y) : màu xanh – (u, v) : màu đỏ – chữ số thể hiện dựa trên màu sắc

Tham khảo cách cài đặt sau:

```
#include <bits/stdc++.h>

using namespace std;
const int N = 1e3 + 5;
int dx[] = { -1, 1, 0, 0 };
int dy[] = { 0, 0, 1, -1 };
int n, m, X, Y, U, V;
int d[N][N]; // d[i][j]=độ dài đường đi từ ngắn nhất (x, y) -> (u, v)
char ch[N][N]; // lưới
struct Item {
    int dist, row, col;
};
priority_queue<Item> q; // hàng đợi có độ ưu tiên - sử dụng CDTL
struct (dist - độ dài đường đi, row - col thể hiện vị trí của ô)
bool operator < (const Item& p1, const Item& p2) {
    return p1.dist > p2.dist; // ưu tiên độ dài nhỏ hơn
}
```

```

bool color[N][N]; // đánh dấu các ô trong lưới, ban đầu khởi tạo
false

bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}

int weight(char c) {
    if (c == 'R' || c == 'G') return 0;
    return c - 48;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> ch[i][j];
            if (ch[i][j] == 'G')
                X = i, Y = j;
            else if (ch[i][j] == 'R')
                U = i, V = j;
            d[i][j] = 10 * (n + m);
        }
    }

    q.push({ 0, X, Y }); // đưa ô (X, Y) vào hàng đợi
    d[X][Y] = 0;
    while (!q.empty()) {
        int x = q.top().row;
        int y = q.top().col;
        q.pop(); // lấy ô (x, y) ra khỏi hàng đợi

        if (color[x][y]) continue;
        color[x][y] = true;
    }
}

```

```

        for (int i = 0; i < 4; i++) {
            int u = x + dx[i], v = y + dy[i]; //xét ô (u,v) kề với (x,y)
            if (!inGrid(u, v) || color[u][v]) continue;
            d[u][v] = min(d[u][v], d[x][y] + weight(ch[u][v]));
            q.push({d[u][v], u, v});
        }
    }
    cout << d[U][V];
}

```

- **Mô phỏng thuật toán Dijkstra trên lưới**



Có thể xem mô phỏng online: [Dijkstra on Grid.gif - Google Drive](#) (File dạng PDF không mô phỏng được)

1.4. Thuật toán BFS 0-1

1.4.1. Giới thiệu thuật toán BFS 0-1

Thuật toán BFS 0-1 là thuật toán tìm các đường đi ngắn nhất từ đỉnh s bắt kè đến các đỉnh còn lại trong đồ thị (có trọng số 0 hoặc 1).

Thuật toán BFS 0-1 có độ phức tạp là $O(N + M)$, trong đó N là số đỉnh, M là số cạnh của đồ thị.

1.4.2. Cấu trúc dữ liệu và giải thuật BFS 0-1

- **Cấu trúc dữ liệu:**

Mảng d : lưu lại giá trị của đường đi ngắn nhất ($d[i]$: giá trị đường đi ngắn nhất $s \rightarrow i$), ban đầu gán bằng ∞ .

Mảng $color$: đánh dấu các đỉnh trong đồ thị, ban đầu tất cả bằng $false$.

Hàng đợi hai đầu q : hàng đợi các đỉnh trong đồ thị.

- **Thuật toán:**

Bước 1: Đưa đỉnh s vào hàng đợi q , gán $d[s] = 0$.

Bước 2: Lấy ra đỉnh u từ hàng đợi q ($u \rightarrow q.front$).

Bước 3: Nếu đỉnh u đã được đánh dấu ($color[u] = true$) thì quay lại bước 2.

Bước 4: Đánh dấu đỉnh u ($color[u] \rightarrow true$). Duyệt các đỉnh v kề với đỉnh u . Nếu đỉnh v chưa được đánh dấu ($color[v] = false$) thì thêm đỉnh v vào trong hàng đợi và gán $d[v] = \min(d[v], d[u] + dist(u, v))$. Việc thêm đỉnh v vào hàng đợi có hai trường hợp xảy ra:

$$\begin{cases} dist(u, v) = 0 \rightarrow q.push_front(v) \\ dist(u, v) = 1 \rightarrow q.push_back(v) \end{cases}$$

Bước 5: Nếu hàng đợi rỗng thì kết thúc thuật toán, ngược lại quay lại bước 2.

1.4.3. Thuật toán BFS 0-1 trên đồ thị lưới

- **Bài toán BFS 0-1 on Grid**

Cho một ma trận lưới $n * m$, các ô vuông có bốn màu màu trắng, đen, xanh lá cây và đỏ. Từ một ô vuông có thể đi sang 4 ô **kề cạnh**. Viết chương trình tìm **số lượng ô đen** ít nhất có thể trên đường đi từ từ (x, y) đến (u, v) .

- (i, j) là ô hàng i cột j .
- Ký tự * là ô màu đen.
- Ký tự . là ô màu trắng.
- Ký tự G là ô (x, y) có màu xanh lá cây.
- Ký tự R là ô (u, v) có màu đỏ.
- Luôn tồn tại đường đi từ (x, y) đến (u, v) .

Dữ liệu

- Dòng đầu tiên: Hai số nguyên dương n và m
- n dòng tiếp theo: Mô tả ma trận lưới.
- Giới hạn: $n, m \leq 1000$

Kết quả

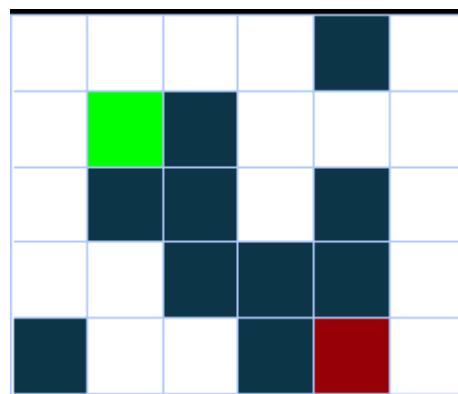
- Kết quả của chương trình.

Ví dụ:

Input	Output
-------	--------

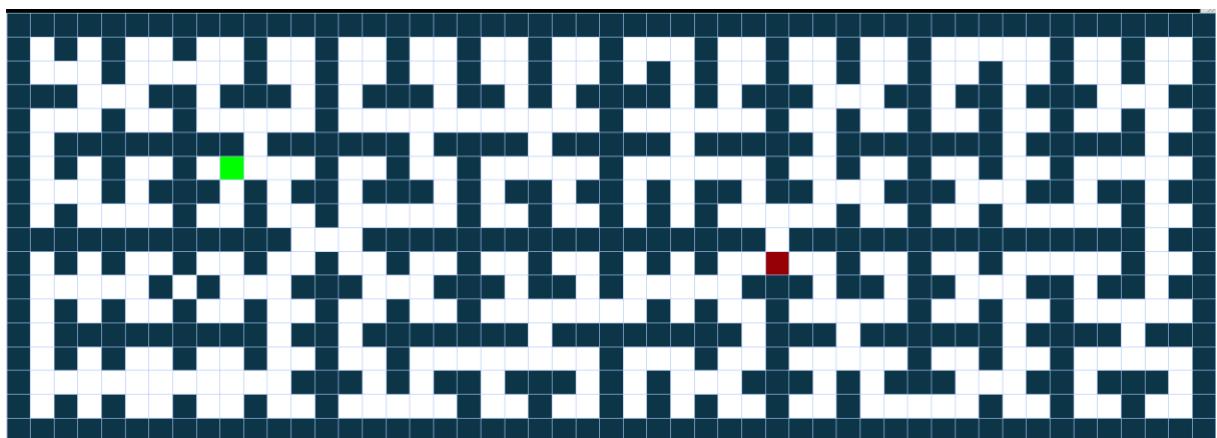
5 6*. .G*** .*. ..***. *..*R.	0
--	---

Giải thích



- **Giải pháp:**

Xem các ô trong lưới như một đỉnh trong đồ thị có hướng. Hai đỉnh có cạnh với nhau nếu chúng kề cạnh, cạnh có trọng số là 1 nếu ô kết thúc là ô đen, ngược lại cạnh có trọng số là 0. Dựa vào cách cài đặt danh sách kề trên lưới và thuật toán BFS 0-1, ta có thể giải quyết bài toán.



(x, y) : MÀU XANH – (u, v) : MÀU ĐỎ

Tham khảo cách cài đặt sau:

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3 + 5;
int dx[] = { -1, 1, 0, 0 };
```

```

int dy[] = { 0, 0, 1, -1 };
int n, m, X, Y, U, V;
int d[N][N]; // d[i][j] = độ dài đường đi từ ngắn nhất (x, y) -> (u, v)

char ch[N][N]; // lưới

deque<pair<int, int>> q; // hàng đợi hai đầu - sử dụng CTDL pair
(first - hàng, second - cột)
bool color[N][N]; // đánh dấu các ô trong lưới, ban đầu khởi tạo
false

bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> ch[i][j];
            if (ch[i][j] == 'G')
                X = i, Y = j;
            else if (ch[i][j] == 'R')
                U = i, V = j;
            d[i][j] = n + m;
        }
    }
    q.push_back({ X, Y }); // đưa ô (X, Y) vào hàng đợi
    d[X][Y] = 0;
    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop_front(); // lấy ô (x, y) ra khỏi hàng đợi
        if (color[x][y]) continue;
        color[x][y] = true;
        for (int i = 0; i < 4; i++) {
            int u = x + dx[i], v = y + dy[i]; // xét ô (u, v) kề với
(x, y)

```

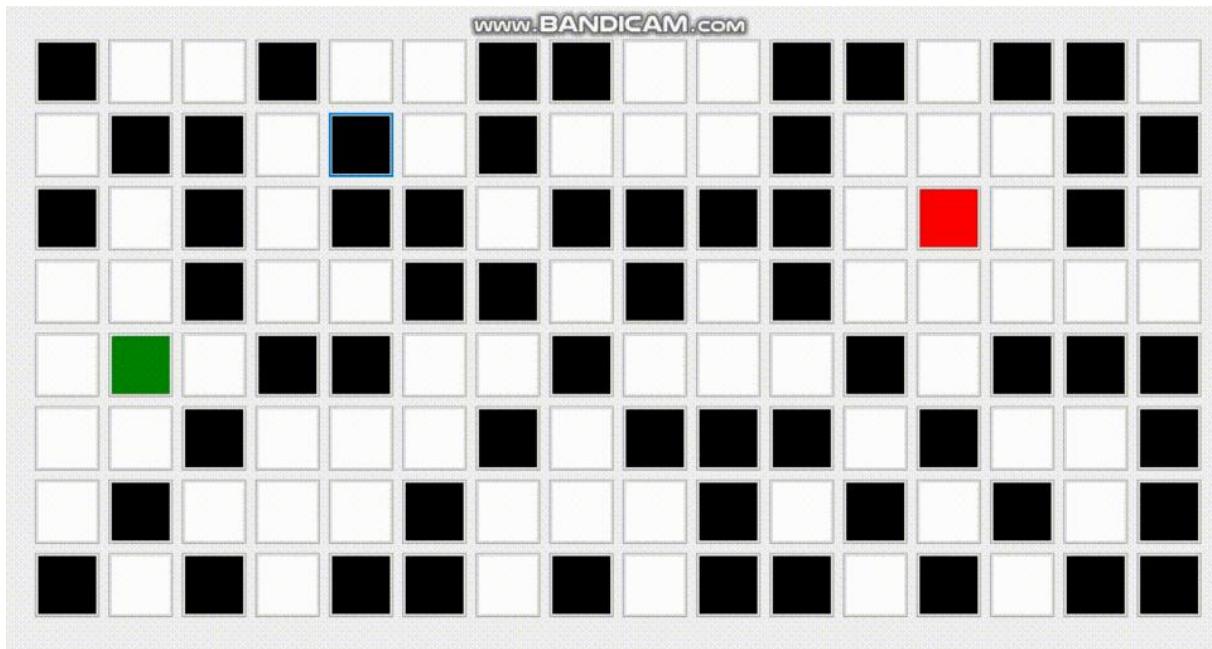
```

        if (!inGrid(u, v) || color[u][v]) continue;
        if (ch[u][v] == '*' ) q.push_back({ u, v });
        else q.push_front({ u, v });
        d[u][v] = min(d[u][v], d[x][y] + (ch[u][v] == '*' ));
    }
}

cout << d[U][V];
}

```

- **Mô phỏng thuật toán BFS 0-1 trên lưới**



Có thể xem mô phỏng online: [BFS 0-1 on Grid.gif - Google Drive](#) (File dạng PDF không mô phỏng được)

1.5. Dial's Algorithm

1.5.1. Giới thiệu thuật toán Dial's

Thuật toán Dial's là thuật toán tìm các đường đi ngắn nhất từ đỉnh s bất kì đến các đỉnh còn lại trong đồ thị (có trọng số không âm và nhỏ hơn hoặc bằng K).

Thuật toán Dial's có độ phức tạp là $O(KN + M)$, trong đó N là số đỉnh, M là số cạnh của đồ thị.

1.5.2. Cấu trúc dữ liệu và giải thuật Dial's

- **Cấu trúc dữ liệu:**

Mảng d : lưu lại giá trị của đường đi ngắn nhất ($d[i]$: giá trị đường đi ngắn nhất $s \rightarrow i$), ban đầu gán bằng ∞ .

Mảng *color*: đánh dấu các đỉnh trong đồ thị, ban đầu tất cả bằng *false*.

$K + 1$ hàng đợi q : hàng đợi các đỉnh trong đồ thị. (mảng hàng đợi)

Con trỏ *ptr*: con trỏ hàng đợi, ban đầu gán bằng 0.

- **Thuật toán:**

Bước 1: Đưa đỉnh s vào hàng đợi $q[0]$, gán $d[s] = 0$.

Bước 2: Trong khi hàng đợi $q[ptr]$ không có đỉnh thì gán $ptr \rightarrow (ptr + 1) \% (K + 1)$

Bước 3: Lấy đỉnh u ra từ hàng đợi $q[ptr]$.

Bước 4: Nếu đỉnh u đã được đánh dấu ($color[u] = true$) thì quay lại bước 2.

Bước 5: Đánh dấu đỉnh u ($color[u] \rightarrow true$). Duyệt các đỉnh v kề với đỉnh u . Nếu đỉnh v chưa được đánh dấu ($color[v] = false$) thì thêm đỉnh v vào trong hàng đợi $q[(ptr + dist(u, v)) \% (K + 1)]$

và gán $d[v] = \min(d[v], d[u] + dist(u, v))$.

Bước 6: Nếu các hàng đợi đều rỗng thì kết thúc thuật toán, ngược lại quay lại bước 2.

1.5.3. Thuật toán Dial's trên đồ thị lưới

Áp dụng thuật toán Dial's cho bài toán ở mục *thuật toán Dijkstra*, tham khảo cách cài đặt sau đây:

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3 + 5;
int dx[] = { -1, 1, 0, 0 };
int dy[] = { 0, 0, 1, -1 };
int n, m, X, Y, U, V;
int d[N][N]; // d[i][j]=độ dài đường đi từ ngắn nhất (x, y) -> (u, v)
char ch[N][N]; // lưới
queue<pair<int, int>> q[10]; // mảng hàng đợi - K = 9
```

```

bool color[N][N]; // đánh dấu các ô trong lưới, ban đầu khởi tạo
false

bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}

int weight(char c) {
    if (c == 'R' || c == 'G') return 0;
    return c - 48;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> ch[i][j];
            if (ch[i][j] == 'G')
                X = i, Y = j;
            else if (ch[i][j] == 'R')
                U = i, V = j;
            d[i][j] = 10 * (n + m);
        }
    }
    q[0].push({X, Y}); // đưa ô (X, Y) vào hàng đợi q[0]
    d[X][Y] = 0;
    int ptr = 0;
    while (true) {
        int cycle = ptr;
        while (q[ptr].empty()) {
            ptr = (ptr + 1) % 10;
            if (ptr == cycle) break;
        }
        if (q[ptr].empty()) break;
        int x = q[ptr].front().first;
        int y = q[ptr].front().second;
        q[ptr].pop(); // lấy ô (x, y) ra khỏi hàng đợi
        if (color[x][y]) continue;
        color[x][y] = true;
    }
}

```

```

    for (int i = 0; i < 4; i++) {
        int u = x + dx[i], v = y + dy[i]; //xét ô (u,v) kề với
        (x,y)
        if (!inGrid(u, v) || color[u][v]) continue;
        d[u][v] = min(d[u][v], d[x][y] + weight(ch[u][v]));
        q[(ptr + weight(ch[u][v])) % 10].push({u, v});
    }
}
cout << d[U][V];
}

```

- **Mô phỏng thuật toán Dial trên lưới**

9	4	0	7	2	3	3	8	2	7	0	7	5	0	5	2
4	3	0	3	3	4	1	3	5	3	6	9	1	0	6	1
1	7	3	3	6	9	4	5	1	0	2	3	3	6	3	
6	6	8	0	5	6	7	0	5	3	4	9	9	4	4	7
2		8	9	3	8	7	2	3	5	9	5	2	0	2	5
0	9	7	0	3	3	0	1	5	4	4	8	7	9	0	2
8	4	3	0	5	4	3	5	9	4	2	5	3	8	2	0
3	5	4	0	2	4	1	1	9	4	4	9	4	9	2	4

Có thể xem mô phỏng online: [Dial's Algorithm.gif - Google Drive](#) (File dạng PDF không mô phỏng được)

2. Ứng dụng

2.1. Ghép chữ (Đề thi DHĐBBB khối 11 năm 2021)

2.1.1. Đề bài

Các bé trường mầm non SuperKids đang chơi trò chơi ghép chữ trên một sân hình chữ nhật kích thước $m \times n$ được chia thành lưới ô vuông đơn vị. Các hàng ô đánh số từ 1 tới m từ trên xuống và các cột ô được đánh số từ 1 tới n từ trái qua phải. Ô nằm trên giao của hàng i và cột j được gọi là ô (i, j) và trên đó chứa đúng một chữ cái hoa a_{ij} .

Mỗi bé khi chơi được cho trước xâu ký tự S độ dài k chỉ gồm các chữ cái hoa. Bé được chọn một ô để xuất phát và thực hiện trò chơi qua nhiều lượt. Tại mỗi lượt, bé bắt buộc phải di chuyển sang một trong bốn ô kề cạnh với ô đang

đứng, sau đó bé được viết ra đúng một chữ cái bằng với chữ cái tại ô vừa tới nếu muốn. Mục đích của bé là viết ra được xâu ký tự S đã cho. Chú ý rằng các chữ cái phải được viết ra lần lượt theo đúng thứ tự trong xâu S và khi tới một ô chỉ được viết ra đúng một chữ cái.

Yêu cầu: Hãy giúp bé thực hiện được mục đích của mình trong trò chơi với số lần di chuyển ít nhất. Cho biết số lần di chuyển theo phương án tìm được

Dữ liệu: Vào từ file văn bản SPELL.INP

- Dòng 1 chứa ba số nguyên dương m, n, k ($2 \leq m, n, k \leq 300$) cách nhau bởi dấu cách
- Dòng 2 chứa xâu ký tự S gồm đúng k chữ cái hoa viết liền.
- m dòng tiếp theo, dòng thứ i chứa n chữ cái hoa liền nhau, chữ cái thứ j là a_{ij}
- Dữ liệu đảm bảo rằng mọi ký tự của xâu S đều có mặt trong ít nhất một ô của sân

Kết quả: Ghi ra file văn bản SPELL.OUT một số nguyên duy nhất là số lần di chuyển ít nhất mà bé cần thực hiện để đạt được mục đích của trò chơi.

SPELL.INP	SPELL.OUT
2 4 6 DHDBBB DHAB ABBD	7

Giải thích:

Bé có thể chọn ô (2, 1) để xuất phát. Sau đó:

(2, 1)
 \downarrow
 $(1, 1) \rightarrow D$
 \downarrow
 $(1, 2) \rightarrow H$
 \downarrow
 $(1, 1) \rightarrow D$
 \downarrow
 $(2, 1)$
 \downarrow
 $(2, 2) \rightarrow B$
 \downarrow

$(2, 3) \rightarrow B$



$(2, 2) \rightarrow B$

Bộ test chia làm 3 subtasks:

- Subtask 1 (30% số điểm): $m, n, k \leq 4$
- Subtask 2 (30% số điểm): $m, n, k \leq 100$
- Subtask 3 (40% số điểm): Không có ràng buộc bổ sung ngoài các ràng buộc đã nêu trong đề bài.

2.1.2. Giải thuật

So với lý thuyết ở trên, cách giải bài tập này có hai điểm mới:

- Ta sẽ BFS theo **trạng thái** của ô trong lưới.
- Duyệt BFS từ nhiều **nguồn**.

Trạng thái của một ô sẽ bao gồm: vị trí hàng (u), vị trí cột (v) và số lượng chữ cái đã viết được (w).

Xem trạng thái của các ô trong lưới như các nút của một đồ thị, từ đồ thị này ta sẽ duyệt BFS để tìm ra kết quả. Trong đó đỉnh (u, v, w) có thể đi đến

$$(u, v, w) \rightarrow \begin{cases} (u - 1, v, w) \\ (u - 1, v, w + 1) & \text{if } a[u - 1][v] = s[w] \\ (u + 1, v, w) \\ (u + 1, v, w + 1) & \text{if } a[u + 1][v] = s[w] \\ (u, v - 1, w) \\ (u, v - 1, w + 1) & \text{if } a[u][v - 1] = s[w] \\ (u, v + 1, w) \\ (u, v + 1, w + 1) & \text{if } a[u][v + 1] = s[w] \end{cases}$$

Lưu ý xâu s được đánh vị trí từ 0.

Đề bài không ràng buộc vị trí ô ban đầu, nên ta sẽ duyệt BFS từ nhiều nguồn.

Cụ thể các nguồn đó là các nút có trạng thái $(i, j, 0)$.

Sau khi duyệt BFS, ta cần độ dài đường đi đến (i, j, k) trong đó k là độ dài xâu s .

Độ phức tạp của giải thuật này là $O(n * m * k)$.

Tuy nhiên độ phức tạp thực sự là $4 * n * m * k = 10^8 + 8 * 10^6$ cho nên ta cần tham lam một chút.

Tham lam như sau: nếu đỉnh (u, v, w) đến được $(u - 1, v, w + 1)$ thì ta sẽ không xét đến đỉnh $(u - 1, v, w)$, tương tự cho ba hướng còn lại.

2.1.2. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
const int N = 300 + 5;
int dx[] = { 0, 0, -1, 1 };
int dy[] = { -1, 1, 0, 0 };

int n, m, k, d[N][N][N];
bool color[N][N][N];
string s;
char a[N][N];
struct State {
    int rowIndex, colIndex, writeCheck;
};

bool inGrid(int r, int c) {
    return (r >= 1 && r <= n && c >= 1 && c <= m) ? true : false;
}
queue<State> Q;
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m >> k >> s;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            Q.push({ i, j, 0 });
            for (int x = 1; x <= k; x++)
                d[i][j][x] = (n + m) * x;
        }
    }
    while (!Q.empty()) {
        auto [r, c, w] = Q.front();
        Q.pop();
        if (color[r][c][w])
            continue;
        color[r][c][w] = true;
        for (int i = 0; i < 4; i++) {
            int nr = r + dx[i], nc = c + dy[i];
            if (inGrid(nr, nc)) {
                if (a[nr][nc] == '#') {
                    if (w + 1 <= k)
                        d[nr][nc][w + 1] += d[r][c][w];
                } else if (a[nr][nc] == 'W') {
                    if (w - 1 >= 1)
                        d[nr][nc][w - 1] += d[r][c][w];
                } else if (a[nr][nc] == 'B') {
                    if (w - 1 >= 1)
                        d[nr][nc][w - 1] += d[r][c][w];
                    if (w + 1 <= k)
                        d[nr][nc][w + 1] += d[r][c][w];
                }
            }
        }
    }
}
```

```

    if (w == k) return cout << d[r][c][w], 0;

    if (color[r][c][w]) continue;
    color[r][c][w] = true;

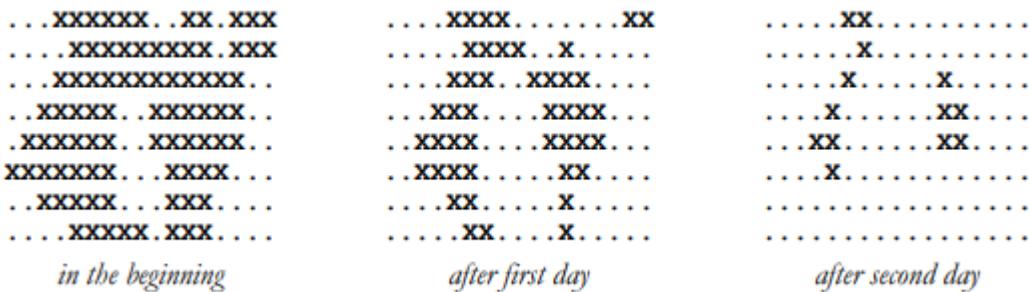
    for (int i = 0; i < 4; i++) {
        int u = r + dx[i]; int v = c + dy[i];
        int t = a[u][v] == s[w] ? w + 1 : w;
        if (!inGrid(u, v) || color[u][v][t]) continue;
        d[u][v][t] = min(d[u][v][t], d[r][c][w] + 1);
        Q.push({u, v, t});
    }
}
}

```

2.2. Hồ thiên nga (COI 2005)

2.2.1. Đề bài

Hai con thiên nga đang ở trong một cái hồ lớn, nhưng chúng lại đang bị chia cắt bởi băng đóng trong hồ nước. Hồ nước có dạng hình chữ nhật được chia thành r dòng c cột. Một số ô trong hồ bị băng đóng. Mùa xuân tới dần, băng trong hồ tan dần – mỗi ngày băng ở tất cả những ô tiếp xúc với nước đang ấm dần trong hồ (tức là kè cạnh một ô không bị đóng băng) sẽ tan ra.



Thiên nga có thể di chuyển tự do ở những ô chứa nước nhưng không thể đi qua những ô bị đóng băng. Bạn hãy tính xem sau bao nhiêu ngày thì đôi thiên nga của chúng ta có thể gặp nhau

Dữ liệu:

- Dòng đầu tiên chứa 2 số r và c , $1 \leq r, c \leq 1500$.
- Mỗi dòng trong r dòng tiếp theo chứa c ký tự mô tả hồ nước tại thời điểm hiện tại: '.' (dot) thể hiện 1 ô chứa nước, 'X' thể hiện 1 ô bị đóng băng, và 'L' thể hiện ô có thiên nga. Có chính xác 2 ô chữ L .

Kết quả

- Một dòng duy nhất chứa số ngày đôi thiên nga có thể gặp nhau.

Ví dụ:

Input	Output
10 2 .L .. XX XX XX XX XX XX XX .. .L	3

2.2.2. Giải thuật

Để giải quyết được bài toán, trước tiên ta cần phải biết được chính xác số ngày băng tan của các ô trong lưới. Thông qua thuật toán BFS, ta duyệt BFS từ các ô nước và hai ô của hai con thiên nga để xác định ngày băng tan và lưu lại mảng d , trong $d[i][j]$ thể hiện ngày đầu tiên ô (i, j) trở thành ô nước.

Khi có được mảng d , ta có thể xác định ngày sớm nhất hai con thiên nga này gặp nhau thông qua tìm kiếm nhị phân, BFS hay DFS cơ bản với độ phức tạp là $O(R * C * \log(R + C))$.

Hoặc có thể áp dụng thuật toán BFS 0-1 để giải quyết với độ phức tạp $O(R * C)$.

Ta có nhận xét sau:

- Đôi thiên nga có thể tự do di chuyển đến khi gặp nhau. Tuy nhiên để gặp nhau ở thời gian sớm nhất, thực chất chỉ cần một con di chuyển, con còn lại đợi con kia di chuyển đến.
- Xét ô (u, v) kè cạnh với ô (i, j)
 - + $\text{abs}(d[u][v] - d[i][j]) \leq 1$.

- + Nếu $d[u][v] = d[i][j] + 1$ thì để đi từ (i,j) đến (u,v) mất 1 ngày, ngược lại mất 0 ngày.

Từ nhận xét này, ta có thể áp dụng thuật toán BFS 0-1 để giải quyết bài toán (duyệt BFS 0-1 từ một con thiên nga bất kỳ).

2.2.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e3 + 5;
int dx[] = {0, 0, -1, 1};
int dy[] = {1, -1, 0, 0};
int n, m, u1, v1, u2, v2, d[N][N], f[N][N];
char a[N][N];

bool inGrid(int r, int c) {
    return (r >= 1 && r <= n && c >= 1 && c <= m) ? true : false;
}

int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            if (a[i][j] == 'L') {
                if (v1 == 0) {
                    u1 = i;
                    v1 = j;
                }
                else {
                    u2 = i;
                    v2 = j;
                }
            }
        }
        d[i][j] = f[i][j] = n + m;
    }
}
```

```

queue<pair<int, int>> Q;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
        if (a[i][j] != 'X')
            Q.push({i, j}),
            d[i][j] = 0;

while (!Q.empty()) {
    auto [u, v] = Q.front();
    Q.pop();
    for (int i = 0; i < 4; i++) {
        auto [s, t] = make_pair(u + dx[i], v + dy[i]);
        if (!inGrid(s, t)) continue;
        if (d[s][t] > d[u][v] + 1) {
            Q.push(make_pair(s, t));
            d[s][t] = d[u][v] + 1;
        }
    }
}

deque<pair<int, int>> DQ;
DQ.push_back({u1, v1});
f[u1][v1] = 0;

while (!DQ.empty()) {
    auto [u, v] = DQ.front();
    DQ.pop_front();

    for (int i = 0; i < 4; i++) {
        auto [s, t] = make_pair(u + dx[i], v + dy[i]);
        if (!inGrid(s, t)) continue;
        int w = d[s][t] > f[u][v];
        if (f[s][t] > f[u][v] + w) {
            f[s][t] = f[u][v] + w;
            if (w) DQ.push_back({s, t}); else DQ.push_front({s, t});
        }
    }
}

cout << f[u2][v2];

```

}

2.3. Covid'19 (Đề thi DHĐBBB 2020)

2.3.1. Đề bài

Dịch Covid-19 đã được kiểm soát, trong gần một tháng qua, tại Việt nam không có ca nhiễm mới trong cộng đồng. Lệnh cách ly đã được nới lỏng, Hồng và các bạn được đi học trở lại, đó cũng là thời điểm thầy Phương mở trại HCC cho các bạn yêu thích lập trình, một hoạt động mà thầy đã duy trì trong nhiều năm qua. Tham gia HCC, Hồng rất thích thú với một bài toán của thầy Phương, bài toán mô phỏng việc di chuyển của virus, cụ thể bài toán như sau: Xét lưới ô vuông thước $m \times n$, các dòng được đánh số từ 1 đến m từ trên xuống, các cột được đánh số từ 1 đến n từ trái sang phải. Ô nằm trên giao của dòng i , cột j được gọi là ô (i,j) và ô này chứa một số nguyên dương $a(i,j)$. Nếu một virus đang ở ô (x,y) , virus có thể thực hiện bước di chuyển sau:

- Virus di chuyển sang ô kè cạnh với ô (x,y) nằm trong lưới, việc di chuyển này mất 1 đơn vị thời gian;
- Virus di chuyển sang ô (u,v) nằm trong lưới nếu $u \times v = a(x,y)$, việc di chuyển này mất 3 đơn vị thời gian.

Bài toán yêu cầu tính thời gian nhỏ nhất để virus di chuyển từ ô (p,q) đến ô (r,s) .

Đây là bài toán khó đối với Hồng nên Hồng nhờ các anh chị tham gia kỳ thi Duyên Hải năm 2020 giải giúp.

Yêu cầu: Cho lưới kích thước $m \times n$ và các số trên lưới. Có h câu hỏi, với câu hỏi thứ k ($k = 1, 2, \dots, h$) cần phải trả lời thời gian nhỏ nhất để virus di chuyển từ ô (p_k, q_k) đến ô (r_k, s_k) là bao nhiêu?

Dữ liệu

- Dòng đầu chứa ba số nguyên dương m, n, h ($h \leq 5$);
- Dòng thứ i ($i = 1, 2, \dots, m$) trong m dòng tiếp theo chứa n số nguyên dương $a(i,1), a(i,2), \dots, a(i,n)$. Các số có giá trị không vượt quá 10^6 .
- Dòng thứ k ($k = 1, 2, \dots, h$) trong h dòng tiếp theo chứa bốn số nguyên p_k, q_k, r_k, s_k .

Kết quả

- Ghi ra h dòng, dòng thứ k ($k = 1, 2, \dots, h$) ghi một số nguyên là thời gian nhỏ nhất để virus di chuyển từ ô (p_k, q_k) đến ô (r_k, s_k) .

Ví dụ:

Input	Output
2 5 2	4
8 6 4 1 1	3
1 1 1 1 1	

1	1	2	5
2	5	1	1

2.3.2. Giải thuật

Một ô (x, y) có thể đi đến các ô

$$(x, y) \rightarrow \begin{cases} (x - 1, y), & cost = 1 \\ (x + 1, y), & cost = 1 \\ (x, y - 1), & cost = 1 \\ (x, y + 1), & cost = 1 \\ (u, v), & cost = 3, \quad if u * v = a_{x,y} \end{cases}$$

Để xây dựng danh sách kè (u, v) với (x, y) như trên, ta chỉ cần lưu các ô (i, j) có tích $i * j$ vào một vector $v[i * j]$.

Ta xây một đồ thị có trọng số và áp dụng thuật toán Dijkstra để giải quyết với với độ phức tạp với mỗi truy vấn là $O(mn * \log(mn))$. Tuy nhiên vì trọng số của các cạnh khá nhỏ nên ta có thể sử dụng thuật toán Dial's hay BFS 1-K để giảm độ phức tạp xuống $O(4 * n * m) \sim O(n * m)$.

2.3.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
int dx[] = {0, 0, -1, 1};
int dy[] = {-1, 1, 0, 0};
int n, m, q;
bool OK[N];
vector<vector<int>> a, d;
vector<pair<int, int>> v[N];
bool inGrid(int r, int c) {
    if (r >= 1 && r <= n && c >= 1 && c <= m) return true;
    return false;
}
void bfs1K(int r1, int c1, int r2, int c2) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            OK[a[i][j]] = false;
            d[i][j] = n + m;
        }
    queue<pair<int, int>> q;
    q.push({r1, c1});
    while (!q.empty()) {
        auto [r, c] = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            int nr = r + dx[i], nc = c + dy[i];
            if (inGrid(nr, nc) && !OK[a[nr][nc]]) {
                OK[a[nr][nc]] = true;
                d[nr][nc] = d[r][c] + 1;
                q.push({nr, nc});
            }
        }
    }
}
```

```

    }

d[r1][c1] = 0;
queue<pair<int,int>> Q[4];
Q[0].push({r1, c1});
int ptr = 0;
while (true) {
    int cycle = ptr;
    while (Q[ptr].empty()) {
        ptr = (ptr + 1) % 4;
        if (ptr == cycle) break;
    }
    if (Q[ptr].empty()) break;
    while (!Q[ptr].empty()) {
        int r = Q[ptr].front().first;
        int c = Q[ptr].front().second;
        Q[ptr].pop();

        if (r == r2 && c == c2) {
            cout << d[r][c] << "\n";
            return;
        }

        for (int i = 0; i < 4; i++) {
            int u = r + dx[i];
            int v = c + dy[i];
            if (!inGrid(u, v)) continue;
            if (d[u][v] > d[r][c] + 1) {
                Q[(ptr + 1) % 4].push({u, v});
                d[u][v] = d[r][c] + 1;
            }
        }
    }

    if (OK[a[r][c]])
        continue;
    OK[a[r][c]] = true;

    for (auto p: v[a[r][c]]) {
        int u = p.first;
        int v = p.second;

```

```

        if (d[u][v] > d[r][c] + 3) {
            Q[(ptr + 3) % 4].push({u, v});
            d[u][v] = d[r][c] + 3;
        }
    }
}
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> m >> q;
    a.assign(n + 1, vector<int> (m + 1));
    d.assign(n + 1, vector<int> (m + 1));
    int ptr = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            v[i * j].push_back({i, j});
            ptr++;
        }
    }

    while (q--) {
        int r1, c1, r2, c2;
        cin >> r1 >> c1 >> r2 >> c2;
        bfs1K(r1, c1, r2, c2);
    }
}

```

2.4. Sabor

2.4.1. Đề bài

Trên trực tọa độ Oxy, có một số tọa độ bị chặn.

Trong một bước, tọa độ (x, y) không bị chặn có thể đi đến bốn tọa độ khác $(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)$ nếu tọa độ đó không bị chặn.

Hỏi có bao nhiêu tọa độ có thể đi đến $(0,0)$ không quá D bước.

Dữ liệu

- Dòng đầu chứa hai số nguyên không âm: n ($n \leq 10^4$) và D ($D \leq 10^7$).
- n dòng tiếp theo, mỗi dòng chứa hai số nguyên x, y thể hiện tọa độ ô bị chặn, giá trị tuyệt đối của x và y nhỏ hơn 10^3 . Dữ liệu đảm bảo ô $(0,0)$ không bị chặn.

Kết quả

- Số lượng tọa độ thỏa mãn đề bài.

Ví dụ:

Input	Output
2 5 2	4
8 6 4 1 1	3
1 1 1 1 1	
1 1 2 5	
2 5 1 1	

2.4.2. Giải thuật

Dữ kiện cho giá trị tuyệt đối của các tọa độ bị chặn nhỏ hơn 10^3 , nên đối với các tọa độ có trị tuyệt đối của hoành độ và tung độ **không quá** 10^3 (vùng duyệt BFS), ta có thể xét bằng cách duyệt BFS từ tọa độ $(0,0)$.

Để xét các tọa độ còn lại, ta có một số nhận xét sau:

- Các tọa độ chưa được xét trong BFS chắc chắn không bị chặn
- $d[i][j]$: số bước ít nhất để $(i,j) \rightarrow (0,0)$.
 - o Đối với (i,j) nằm trong vùng duyệt BFS thì $d[i][j]$ đã được xác định qua duyệt BFS.
 - o Đối với (i,j) không nằm trong vùng duyệt BFS, ta có thể xác định như sau:
 - Nếu $i \geq 1000$ thì $d[i][j] = d[i+1][j] + 1$.
 - Nếu $i \leq -1000$ thì $d[i-1][j] = d[i][j] + 1$.
 - Nếu $j \geq 1000$ thì $d[i][j+1] = d[i][j] + 1$.
 - Nếu $j \leq -1000$ thì $d[i][j-1] = d[i][j] + 1$.
 - o Từ đó, đối với (i,j) bất kỳ không nằm trong vùng BFS, ta có thể biết được $d[i][j]$ thông các tọa độ biên của vùng duyệt BFS.

Độ phức tạp: $O(10^6)$.

2.4.3. Cài đặt

Tịnh tiến hệ tọa độ theo vector $(1000, 1000)$ để việc lưu trữ $d[i][j]$ dễ dàng hơn. Vậy thì vùng duyệt BFS sẽ là $(0 \dots 2000, 0 \dots 2000)$.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
```

```

const int N = 2000 + 1;
int dx[] = { 0, 0, -1, 1 };
int dy[] = { -1, 1, 0, 0 };
int n, D, ans, d[N][N];
bool color[N][N];
bool inGrid(int r, int c) {
    return (r >= 0 && r <= 2000 && c >= 0 && c <= 2000) ? true :
false;
}

int path(int n) {
    if (n <= 1) return 0;
    return n * (n - 1) / 2;
}
queue<pair<int, int>> q;
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> D;
    for (int i = 1; i <= n; i++) {
        int x, y;
        cin >> x >> y;
        x += 1000; y += 1000;
        color[x][y] = true;
    }
    for (int i = 0; i <= 2000; i++)
        for (int j = 0; j <= 2000; j++)
            d[i][j] = D + 1;
    q.push({1000, 1000});
    d[1000][1000] = 0;

    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop();

        if (d[x][y] > D) break;
        if (color[x][y]) continue;
    }
}

```

```

        color[x][y] = true;
        ans++;

        for (int i = 0; i < 4; i++) {
            int u = x + dx[i], v = y + dy[i];
            if (!inGrid(u, v) || color[u][v]) continue;
            d[u][v] = min(d[u][v], d[x][y] + 1);
            q.push({u, v});
        }
    }

    for (int i = 0; i <= 2000; i++) {
        ans += max(0ll, D - d[0][i]) + max(0ll, D - d[2000][i]);
        ans += max(0ll, D - d[i][2000]) + max(0ll, D - d[i][0]);
    }

    ans += max(0ll, path(D - d[0][0]));
    ans += max(0ll, path(D - d[0][2000]));
    ans += max(0ll, path(D - d[2000][2000]));
    ans += max(0ll, path(D - d[2000][0]));

    cout << ans;
}

}

```

2.5. Robot

2.5.1. Đề bài (Đề bài của thầy Nguyễn Đức Nghĩa)

Cho lưới ô vuông kích thước n dòng và n cột. Các dòng của lưới được đánh số từ 1 đến n . Các cột của lưới cũng được đánh số từ 1 đến n . Ô nằm trên giao của dòng i và cột j của lưới được gọi là ô (i, j) và (i, j) được gọi là tọa độ của nó. Mỗi ô của lưới chứa một số thuộc tập $\{0, 1\}$. Ô chứa số 0 được gọi là ô tự do còn ô chứa số 1 được gọi là ô bị cản. Robot được đặt ở ô (L_1, C_1) cần phải di chuyển đến ô (L_2, C_2) . Robot chỉ có thể di chuyển theo hướng thẳng đứng hoặc hướng nằm ngang.

Yêu cầu: Cần xác định:

- Số lần đổi hướng ít nhất để robot có thể di chuyển từ ô (L_1, C_1) tới ô (L_2, C_2)
- Số lần đổi hướng ít nhất để robot có thể di chuyển từ ô (L_1, C_1) tới ô (L_2, C_2) trong tình huống được phép biến một ô bị cản thành ô tự do.
- Số lượng các ô bị cản mà việc loại bỏ bất cứ một ô nào trong số chúng, ta đều đạt được số lần đổi hướng như trong câu 2).

Dữ liệu

- Dòng thứ nhất chứa số nguyên n ($< n < 1000$);
- n dòng tiếp mỗi dòng chứa n số 0 hoặc 1 được ghi cách nhau bởi dấu cách mô tả trạng thái của lưới;
- Dòng thứ $n + 2$ chứa 4 số L_1, C_1, L_2, C_2 (đảm bảo là các ô (L_1, C_1) và (L_2, C_2) là các ô tự do).

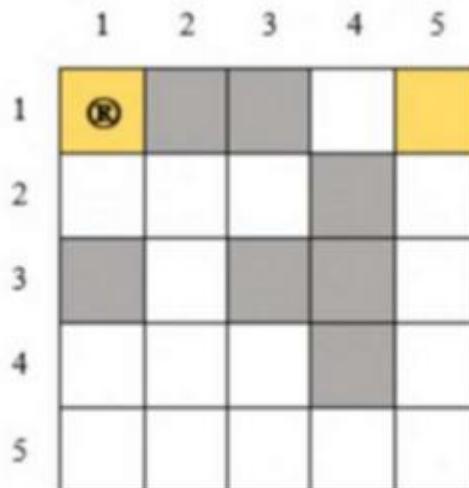
Kết quả

- Ghi ra ba số nguyên là các câu trả lời cho 3 yêu cầu tương ứng nêu trong đầu bài.

Ví dụ:

Input	Output
5 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 5	4 2 2

Giải thích ví dụ



Các ô bị cản cần tìm trong yêu cầu 3) là (2,4) và (3,1)

2.5.1. Giải thuật

Ý tưởng chính của bài toán này là

- Trạng thái của một ô (hàng, cột và hướng đi)
- Duyệt BFS 0-1 trạng thái từ (l_1, c_1) và từ (l_2, c_2)

Xây dựng đồ thị dựa trên trạng thái của các ô trong lưới, cụ thể (x, y, w) (lần lượt là hàng cột, và hướng đi)

$$w = \begin{cases} 0, & up \\ 1, & left \\ 2, & right \\ 3, & down \end{cases}$$

$$(x, y, w) \rightarrow \begin{cases} (x', y', w) & cost = 0 \\ (x, y, w') & cost = 1 \end{cases}$$

Với (x', y') là vị trí ô (x, y) khi đi theo hướng w , (x, y, w') nghĩa là thay đổi hướng đi của robot.

Sau đó duyệt BFS 0-1 hai lần, từ (l_1, c_1) và từ (l_2, c_2) . Lưu lại mảng $d_1[i][j][k]$, với ý nghĩa là số lần chuyển hướng ít nhất để robot ở $(l_1, c_1, 0..3) \rightarrow (i, j)$ mà không phải đi qua bất kỳ ô bị cản nào ngoài ô (i, j) . Tương tự với $d2[i][j][k]$, robot bắt đầu ở (l_2, c_2) .

Câu truy vấn nhất: $\min(d_1[l_2][c_2][i])$ hoặc $\min(d_2[l_1][c_1][i])$ với $i = 0 \dots 3$

Câu thứ vấn hai: $\min(d_1[i][j][w] + d_2[i][j][3 - w])$ với $a[i][j] = 1$, $w = 0 \dots 3$ ($3 - w$ là hướng ngược với w).

Câu truy vấn ba: áp dụng giống như câu truy vấn hai

Tuy nhiên ở truy vấn hai và ba, cần lưu ý trường hợp khi không cần bỏ ô bị cản nào robot vẫn di chuyển với ít lần thay đổi hướng nhất.

2.5.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1000 + 1;
int dx[] = { -1, 0, 0, 1 };
int dy[] = { 0, -1, 1, 0 };

int n, l1, c1, l2, c2;
bool a[N][N];
int d1[N][N][4], d2[N][N][4];

struct State {
    int x, y, z;
};

bool inGrid(int x, int y) {
    return (x >= 1 && x <= n && y >= 1 && y <= n);
}
```

```

void bfs(int X, int Y, int d[][][N][4]) {
    deque<State> dq;

    for (int i = 0; i < 4; i++) {
        dq.push_back({X, Y, i});
        d[X][Y][i] = 0;
    }

    while (!dq.empty()) {
        int x = dq.front().x;
        int y = dq.front().y;
        int z = dq.front().z;
        dq.pop_front();

        if (!a[x][y]) {
            int u = x + dx[z], v = y + dy[z];
            if (inGrid(u, v) && d[u][v][z] > d[x][y][z]) {
                d[u][v][z] = d[x][y][z];
                dq.push_front({u, v, z});
            }
        }
    }

    for (int i = 0; i < 4; i++) if (i != z) {
        if (d[x][y][i] > d[x][y][z] + 1) {
            d[x][y][i] = d[x][y][z] + 1;
            dq.push_back({x, y, i});
        }
    }
}

int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) {
            cin >> a[i][j];
}

```

```

        for (int k = 0; k < 4; k++)
            d1[i][j][k] = d2[i][j][k] = 4 * n * n;
    }

cin >> l1 >> c1 >> l2 >> c2;
bfs(l1, c1, d1);
bfs(l2, c2, d2);

int ans1 = 4 * n * n, ans2 = 4 * n * n, ans3 = 0;
for (int i = 0; i < 4; i++) {
    ans1 = min(ans1, d1[l2][c2][i]);
}

for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        if (a[i][j]) {
            int tmp = 4 * n * n;
            for (int k = 0; k < 4; k++)
                tmp = min(tmp, d1[i][j][k] + d2[i][j][3 - k]);
            if (ans2 > tmp) {
                ans2 = tmp;
                ans3 = 1;
            }
            else if (ans2 == tmp) {
                ans3++;
            }
        }
    if (ans2 >= ans1) {
        ans3 = 0;
        ans2 = ans1;
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                ans3 += a[i][j];
    }
    cout << ans1 << " " << ans2 << " " << ans3;
}

```

2.6. Thuật toán Dial trên lưới

2.6.1. Đề bài

Cho một lưới gồm m hàng, n cột. Bạn Hiếu có k lọ sơn, và với mỗi ô, bạn sẽ chọn một trong số k màu để tô lên.

Từ một ô có thể đi sang các ô kề cạnh. Cụ thể hơn, từ ô (x, y) , chúng ta có thể đi sang ô $(x - 1, y), (x, y - 1), (x, y + 1), (x + 1, y)$.

Để đi từ ô có màu x sang ô có màu y , chi phí dịch chuyển là c_{xy} . Lưu ý, c_{xy} có thể khác c_{yx} .

Hãy trả lời q câu hỏi, mỗi câu có dạng "Chi phí thấp nhất để đi từ ô (u, v) tới ô (x, y) là bao nhiêu?".

Dữ liệu

- Dòng đầu chứa 3 số m, n, k
- m dòng sau, dòng thứ i chứa n số: $a_{i1}, a_{i2}, \dots, a_{in}$ biểu thị màu của từng ô.
- k dòng sau, dòng thứ i chứa k số: $c_{i1}, c_{i2}, \dots, c_{ik}$. c_{ij} là chi phí chuyển từ ô màu i sang ô màu j .
- Dòng tiếp theo chứa số q
- q dòng cuối, dòng thứ i chứa 4 số u_i, v_i, x_i, y_i biểu thị cho câu hỏi thứ i .

Kết quả: In ra q dòng, mỗi dòng in ra đáp án của câu hỏi thứ i .

Ví dụ:

Input	Output
3 4 2	2
2 1 1 2	5
1 2 1 2	2
1 1 2 2	
1 2	
3 1	
3	
1 2 2 3	
1 1 3 2	
2 1 2 2	

Giới hạn

- $m, n \leq 2000, k \leq 4, q \leq 10$
- $\forall 1 \leq i \leq m, 1 \leq j \leq n: a_{ij} \leq 4$
- $\forall 1 \leq i, j \leq k: c_{ij} \leq 10. \forall 1 \leq i \leq k: c_{ii} = 1$

2.6.2. Giải thuật

Theo như đề bài, chúng ta phải tìm đường đi ngắn nhất từ đỉnh $(u, v) \rightarrow (x, y)$.

Theo lý thuyết ở trên, chúng ta có thể sử dụng thuật toán Dijkstra kết hợp sử dụng cấu trúc dữ liệu heap để làm bài này.

Tuy nhiên, vì chi phí chuyển giữa hai cạnh ≤ 10 , nên chúng ta sẽ sử dụng Dial thay cho heap.

2.6.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e3 + 1;
int m, n, k, q, oo;
int a[N][N], c[N][N], dist[N][N];
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, 1, -1};

struct Cell {
    int row, col;
};

bool inGrid(int x, int y) {
    return (x >= 1 && x <= m && y >= 1 && y <= n);
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> m >> n >> k;
    oo = 10 * (n + m);

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            cin >> a[i][j];
        }
    }

    for (int i = 1; i <= k; i++) {
        for (int j = 1; j <= k; j++) {
            cin >> c[i][j];
        }
    }

    cin >> q;
    while (q--) {
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                dist[i][j] = oo;
            }
        }

        int x, y, u, v; cin >> x >> y >> u >> v;
        queue<Cell> q[11];
        int ptr = 0;
        dist[x][y] = 0;
        q[0].push({x, y});
        while (true) {
            int cycle = ptr;
            while (q[ptr].empty()) {
                ptr = (ptr + 1) % 11;
                if (ptr == cycle) break;
            }

            Cell cur = q[ptr].front();
            q[ptr].pop();

            for (int i = 0; i < 4; i++) {
                int nx = cur.row + dx[i];
                int ny = cur.col + dy[i];

                if (inGrid(nx, ny) && dist[nx][ny] > dist[cur.row][cur.col]) {
                    dist[nx][ny] = dist[cur.row][cur.col];
                    q[ptr].push({nx, ny});
                }
            }
        }
    }
}
```

```

        }
        if (q[ptr].empty()) break;

        Cell cell = q[ptr].front(); q[ptr].pop();
        int x = cell.row, y = cell.col;
        if (x == u && y == v) break;
        for (int i = 0; i < 4; i++) {
            int u = x + dx[i], v = y + dy[i], w =
            c[a[x][y]][a[u][v]];
            if (inGrid(u, v) && dist[u][v] > dist[x][y] + w) {
                dist[u][v] = dist[x][y] + w;
                q[(ptr + w) % 11].push({u, v});
            }
        }
        cout << dist[u][v] << "\n";
    }
}

```

2.7. Super Computer

2.7.1. Đề bài (Đề bài của thày Đỗ Đức Đông)

Với 1,57 triệu bộ vi xử lý và có tốc độ xử lý 16.32 triệu tỷ phép tính/giây, cỗ máy Sequoia của IBM đã trở thành siêu máy tính (supercomputer) mạnh nhất thế giới.

Được sử dụng cỗ máy Sequoia và muốn chứng minh sức mạnh của máy có thể thao tác với số cực lớn, Bờm đã lập trình một trò chơi tìm đường đi trên bảng số như sau:

Xét bảng số có kích thước $m \times n$, các hàng của bảng được đánh số từ trên xuống dưới bắt đầu từ 1 đến m , còn các cột được đánh số từ trái sang phải, bắt đầu từ 1 đến n , ô nằm ở vị trí hàng i và cột j gọi là ô (i, j) và chứa giá trị C_{ij} ($0 < C_{ij} \leq 10^9$). Người chơi cần tìm một đường đi từ ô $(1,1)$ và kết thúc tại ô (m, n) theo quy tắc: tại mỗi ô, người chơi có thể di chuyển sang các ô chung cạnh với ô đó và không được di ra ngoài bảng. Việc đánh giá cho điểm một đường đi được dựa trên số lượng số 0 liên tiếp nằm ở cuối của tích các số trong các ô thuộc đường đi, số lượng số 0 càng ít thì càng được đánh giá cao.

Yêu cầu: Cho bảng số, tìm đường đi có số lượng số 0 liên tiếp nằm ở cuối của tích các số trong ô thuộc đường đi là ít nhất.

Dữ liệu: Vào từ file văn bản SC.INP có dạng:

- Dòng đầu tiên chứa hai số nguyên m, n ($m, n \leq 5000$),
- m dòng tiếp theo, mỗi dòng n số nguyên dương mô tả bảng số.

Kết quả: Đưa ra file văn bản SC.OUT một số nguyên là số lượng số 0 liên tiếp nằm ở cuối của tích các số trong các ô thuộc đường đi tìm được.

Ví dụ:

SC . INP	SC . OUT
2 3 2 5 5 5 1 2	1
5 3 1 1 10 10 1 100 1 1 1000 1 10 10000 1 1 1	0

Chú ý: 60% số test ứng với 60% số điểm có $m, n \leq 50$ và c_{ij} có dạng 10^k ($k \leq 9$)

2.7.2. Giải thuật

Subtask 1: $c_i = 10^k$ nên mỗi số trên bảng đóng góp k chữ số 0 vào tích. Ta thực hiện việc tìm đường đi ngắn nhất trên lưới. Chi phí được định nghĩa bằng tổng k trên đường đi. Vì m, n bé nên có thể cài Dijkstra $O(V^2)$ trong đó V là số đỉnh của đồ thị.

Subtask 2:

Xét số nguyên dương T bất kỳ. Phân tích T thành $T = y * 10^x$, trong đó x lớn nhất có thể (tức y NTCN với 10). Khi đó số chữ số 0 tận cùng của số nguyên dương T này là x .

Nhìn kĩ hơn, ta có $T = y * 2^x * 5^x = yy * 2^{x+a} * 5^{x+b}$ (a, b là số mũ của 2 và 5 trong $y \Rightarrow yy$ NTCN với 2, với 5) ($a = 0$ hoặc $b = 0$). Như vậy, với mỗi đường đi trên bảng thì số chữ số 0 tận cùng là $\min(\text{số mũ của } 2, \text{số mũ của } 5)$ trong tích các số trên đường đi. Ta có $a^b * a^c = a^{b+c}$ nên thực chất, số mũ của tích chính là tổng số mũ của từng nhân tử.

Đến đây các bạn đã hình dung được yêu cầu của bài toán. Cách làm tìm đường đi ngắn nhất theo pair<int,int> : <tổng mũ 2, tổng mũ 5> là **SAI**.

Cách thức hợp lý để giải quyết vấn đề này như sau : Ta tìm hai đường đi, một đường có tổng số mũ của 2 bé nhất, một đường có tổng số mũ của 5 bé nhất. Giả sử tổng số mũ 2 bé nhất là $cnt2$, tổng số mũ 5 bé nhất là $cnt5$, thì đáp án bài toán là $\min(cnt2, cnt5)$

Vì sao lại như vậy? Gọi đường đi có tổng mũ 2 bé nhất là “đường đi mũ 2”. Tương tự cho “đường đi mũ 5”. “Đường đi mũ 2” có tổng mũ 5 là $c5$, “đường đi mũ 5” có tổng mũ 2 là $c2$. Vậy phải có $c5 \geq cnt5$ và $c2 \geq cnt2$. Không mất tính tổng quát, ta giả sử $cnt2 \leq cnt5$. Thì $cnt2 \leq cnt5 \leq c5$, do đó chi phí “đường đi mũ 2” này là $\min(cnt2, c5) = cnt2$. Trong trường hợp giả sử này ta

cũng có $c2 \geq cnt2$. Chi phí “đường đi mũ 5” lúc này là $\min(cnt5, c2) \geq \min(cnt5, cnt2) = cnt2$. Trường hợp $cnt2 > cnt5$ hoàn toàn tương tự.

Về mặt cài đặt, có thể dùng Dijkstra Heap. Độ phức tạp là $O(m * n * \log_2(m * n))$

2.7.3. Cài đặt

```
#include<bits/stdc++.h>
using namespace std;

const int INF = 1e9;
const int MAXN = 505;
int nrows, ncols;
int val[MAXN][MAXN], weight[MAXN][MAXN];

struct State {
    int x,y,weight;
    State(int x, int y, int weight):x(x), y(y), weight(weight) {}
    bool operator< (const State& rhs) const {
        return weight > rhs.weight;
    }
};
int x_dir[4] = {-1,+1, 0, 0};
int y_dir[4] = { 0, 0,-1,+1};
bool is_valid(int x, int y) {
    return 0 < min(x,y) and x <= nrows and y <= ncols; }
int dist[MAXN][MAXN];
int dijkstra(int base) {
    for (int r = 1; r <= nrows; r++)
        for (int c = 1; c <= ncols; c++) {
            int tmp = val[r][c];
            weight[r][c] = 0;
            for(;tmp%base == 0;tmp /= base) ++weight[r][c];
            dist[r][c] = INF;
        }
    priority_queue<State> heap;
    heap.push(State(1,1, dist[1][1] = weight[1][1]));
    while (!heap.empty()) {
        State cur = heap.top(); heap.pop();
        if (cur.weight != dist[cur.x][cur.y]) continue;
        for (int d = 0; d < 4; d++) {
            int u = cur.x + x_dir[d], v = cur.y + y_dir[d];
            if (!is_valid(u,v)) continue;
            int relax = cur.weight + weight[u][v];
            if (relax < dist[u][v])
                dist[u][v] = relax,
                heap.push(State(u,v,relax));
        }
    }
    return dist[nrows][ncols];
}

int main()
{
    scanf("%d %d", &nrows, &ncols);
    for (int r = 1; r <= nrows; r++)
        for (int c = 1; c <= ncols; c++)
            val[r][c] = (rand() % 5) + 1;
}
```

```

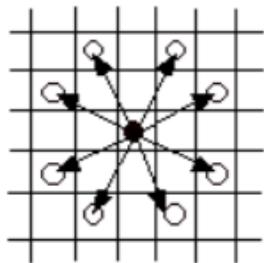
    for (int c = 1; c <= ncols; c++) scanf("%d",
&val[r][c]);
    printf("%d", min(dijkstra(2), dijkstra(5)));
}

```

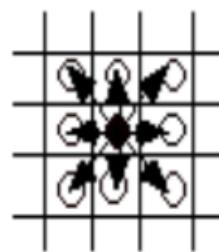
2.8. Camelot

2.8.1. Đề bài (Đề bài của thầy Đỗ Đức Đông)

Vua Arthur và các hiệp sĩ bàn tròn thường gặp nhau vào đầu năm mới để ăn mừng tình bạn của họ. Để tưởng nhớ sự kiện này, chúng ta xem xét một trò chơi, trong đó có một quân vua và một vài quân mã được đặt ngẫu nhiên trên các ô riêng biệt. Bàn cờ có kích thước $n \times n$, trên bàn cờ có một số ô cấm những ô còn lại là những ô tự do – ô có thể di chuyển vào được. Các ô đặt quân mã và quân vua đang đứng ở các ô tự do.



Quy tắc di chuyển của quân mã



Quy tắc di chuyển của quân vua

Quy tắc di chuyển của quân mã Quy tắc di chuyển của quân vua Tại mỗi bước tất cả các quân đều phải di chuyển theo quy tắc và không được đi vào ô cấm, hãy tìm cách di chuyển để chúng gặp nhau nhanh nhất.

Dữ liệu

- Dòng đầu là số n ;
- n dòng tiếp theo, mỗi dòng 1 xâu n ký tự, gồm các ký tự \square thể hiện ô trống, $\#$ thể hiện ô cấm không được phép đi vào, T thể hiện vị trí vua đang đứng, M thể hiện vị trí quân mã đang đứng.

Output

- Gồm một số là số bước ít nhất để các quân gặp nhau, nếu không thể gặp được nhau ghi -1 .

Ví dụ:

Input	Output
5 M.....#... ..#..# ...#T	2

Ràng buộc

- Subtask 1: $n \leq 20$ và chỉ có một quân mã;
- Subtask 2: $n \leq 100$ và chỉ có một quân mã;
- Subtask 3: $n \leq 100$.

2.8.2. Giải thuật

Subtask 1 :

$n \leq 20$, rất bé nên ta có thể làm như sau: Coi mỗi một trạng thái gồm vị trí hiện của quân vua (x, y) và quân mã (u, v) như là một đỉnh của đồ thị (x, y, u, v) . Mỗi khi hai quân cờ này di chuyển tới (x', y') và (u', v') , ta có được trạng thái mới là (x', y', u', v') , và như vậy coi như là đã đi tới một đỉnh khác. Cần tìm đường đi ngắn nhất từ đỉnh gốc tới mọi đỉnh (x, y, x, y) .

Subtask 2&3 :

Nhận xét : Giả sử một con mã đi tới ô (x, y) được trong thời gian ngắn nhất là k bước, thì mọi cách đi tới ô (x, y) này đều phải tồn số bước $k' \geq k$ có cùng tính chẵn lẻ với k . Với con vua cũng có tính chất tương tự. Từ vị trí bắt đầu tới ví trí kết thúc cố định, thì mọi cách đi đều tồn chẵn bước hoặc lẻ bước.

Chứng minh : Mọi chu trình trên tất cả đường đi có thể của quân mã hoặc vua đều có chẵn bước. Sau mỗi bước, luôn tồn tại tối thiểu một tọa độ x hoặc một tọa độ y bị đổi tính chẵn lẻ (+1 hoặc -1). Do đó, mỗi nước đi bất kì của quân mã hoặc quân vua cần có thêm một nước đi khác tương ứng để đưa tọa độ x, y của nó về đúng tính chẵn lẻ ban đầu. Quân cờ sẽ tìm cách đi ngắn nhất tới ô đích trong k bước, rồi lại đi theo chu trình để kéo dài thời gian, đợi các quân cờ khác đến cùng lúc với nó.

Từ đây ta có thuật toán. Với mỗi quân cờ, ta sẽ BFS để tính khoảng cách ngắn nhất tới mọi ô trong bàn cờ. Khi xét kết quả, ta duyệt qua từng ô và xem xét ô nào mà mọi quân cờ có thể đi tới được, với tất cả độ dài đường đi có cùng tính chẵn lẻ. Thời gian của quân cờ tới chậm nhất chính là thời gian để mọi quân cờ tập trung tại ô này.

Thuật toán này tổng quát và có thể giải cho bàn cờ có nhiều quân vua và mã.

2.8.3. Cài đặt

Ta dùng mảng $ans[r][c][0/1]$ để lưu độ dài dài đường đi dài nhất chia 2 dư 0 hoặc 1 và kết thúc tại ô (r, c) . Mảng $cnt[r][c][0/1]$ đếm số lượng đường đi như vậy.

```
#include<bits/stdc++.h>
using namespace std;

typedef long long Int;
typedef long double Real;

const int MOD = 2004010501; //MOD > 2e9

const int MAX = 105;
int siz;
char board[MAX][MAX];
```

```

bool is_valid(const int& x, const int& y) {
    return (1 <= x and x <= siz) and (1 <= y and y <= siz);
}

const int K_xmove[8] = {-1,-1,-1, 0, 0,+1,+1,+1};
const int K_ymove[8] = {-1, 0,+1,-1,+1,-1, 0,+1};

const int N_xmove[8] = {-2,-2,-1,-1,+1,+1,+2,+2};
const int N_ymove[8] = {-1,+1,-2,+2,-2,+2,-1,+1};

const int INF = 1e9;
int ans[MAX][MAX][2], cnt[MAX][MAX][2];
void maximize(int& var, const int& val) { if (val > var) var = val;
}

struct Cell {
    int x,y;
    Cell(int x = 0, int y = 0) : x(x), y(y) {}
};

int num_pieces = 0;
int dist[MAX][MAX];
void find_path(int sx, int sy, bool is_king = false) {
    ++num_pieces;
    for (int r = 1; r <= siz; r++)
        for (int c = 1; c <= siz; c++) dist[r][c] = INF;
    dist[sx][sy] = 0;
    queue<Cell> bfs;
    bfs.push(Cell(sx,sy));
    while (!bfs.empty()) {
        Cell cur = bfs.front(); bfs.pop();
        maximize(ans[cur.x][cur.y][dist[cur.x][cur.y] % 2],
dist[cur.x][cur.y]);
        ++cnt[cur.x][cur.y][dist[cur.x][cur.y] % 2];
        for (int d = 0; d < 8; d++) {
            int nx = cur.x + ((is_king) ? K_xmove[d] : N_xmove[d]);
            int ny = cur.y + ((is_king) ? K_ymove[d] : N_ymove[d]);
            if (!is_valid(nx,ny)) continue;
            if (dist[nx][ny] < INF) continue;
            if (board[nx][ny] == '#') continue;
            dist[nx][ny] = dist[cur.x][cur.y] + 1;
            bfs.push(Cell(nx,ny));
        }
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> siz;
    for (int i = 1; i <= siz; i++)
        for (int j = 1; j <= siz; j++) cin >> board[i][j];

    for (int i = 1; i <= siz; i++)

```

```

        for (int j = 1; j <= siz; j++) {
            if (board[i][j] == 'M') find_path(i, j, false);
            if (board[i][j] == 'T') find_path(i, j, true);
        }

        int res = INF;
        for (int i = 1; i <= siz; i++)
            for (int j = 1; j <= siz; j++)
                for (int r = 0; r < 2; r++)
                    if (cnt[i][j][r] == num_pieces) res =
min(res, ans[i][j][r]);

        if (res >= INF) res = -1;
        cout << res;
    }
}

```

2.9. Battle of Hogwarts

2.9.1. Đề bài (The 2020 ICPC Vietnam National)

<https://open.kattis.com/problems/battleofhogwarts>

Kẻ thù đang đến Trường Phù thủy và Pháp sư Hogwarts. Hãy giúp Harry và những người bạn bảo vệ ngôi trường!

Hogwarts có thể được xem như một lưới với các r hàng và c cột. Các hàng được đánh số từ 1 đến r từ trên xuống dưới và các cột được đánh số từ 1 đến c từ trái sang phải. Ô ở hàng thứ i và cột thứ j được ký hiệu là (i, j) .

Có 3 loại ô:

- Tường: Kẻ thù không thể đi vào các ô này.
- Bình thường: Kẻ thù có thể đi vào các ô này, nhưng Harry có thể sử dụng phép thuật để ngăn chặn những ô này.
- Dịch thuật : Kẻ thù có thể đi vào các ô này. Nhưng Harry không thể chặn chúng.

Kẻ thù đang tiến vào từ ô $(1, 1)$. Chúng có thể di chuyển giữa 2 ô nếu chúng kề cạnh nhau. Tuy nhiên, chúng không thể di chuyển đến các ô tường hoặc các ô bình thường bị chặn bởi Harry.

Harry phải ngăn chặn kẻ thù đến ô (r, c) . Để làm điều này, anh ta có thể sử dụng phép thuật để chặn một số ô bình thường. Lưu ý rằng nếu ô $(1, 1)$ hoặc ô (r, c) là tường hoặc bị chặn bởi Harry, điều đó có nghĩa là kẻ thù không thể đi từ ô $(1, 1)$ đến ô (r, c) .

Tuy nhiên, thời gian không còn nhiều! Harry cần biết số lượng ô bình thường tối thiểu mà anh ta cần chặn là bao nhiêu. Làm ơn hãy giúp anh ấy!

Dữ liệu

Đầu vào chứa nhiều test. Mỗi test bao gồm:

- Dòng đầu tiên chứa hai số nguyên dương r và c ($1 \leq r * c \leq 10^6$).

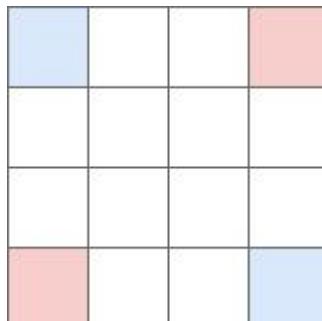
- Trong r dòng tiếp theo, dòng thứ i chứa chính xác c ký tự. Mỗi ký tự có thể là một trong những ký tự sau:
 - # : ô tường
 - . : ô bình thường
 - @ : ô Dịch thuật
- Đầu vào kết thúc bằng hai số 0.

Kết quả

- Đối với mỗi test, in chính xác một dòng duy nhất chứa số ô bình thường tối thiểu mà Harry cần chặn để ngăn kẻ thù tiếp cận ô (r, c) . Nếu không thể, in ra -1 -1.

Ví dụ:

Input	Output
4 4	2
@...#	0
....	
....	
#..@	
4 4	
@...#	
..#. .	
.#. ..	
#..@	
0 0	



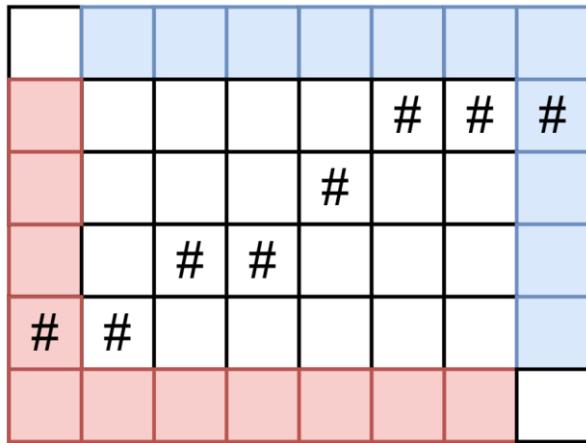
Trong hình trên, các bức tường có màu đỏ, các tế bào bình thường có màu trắng và các tế bào dịch thuật có màu xanh lam.

Harry có thể chặn 2 ô (2,3) và (3,2)..

2.9.2. Giải thuật

Điều kiện để (1,1) không thể đến (r, c) đó là

- Một trong hai ô $(1,1)$ và ô (r,c) là ô tường.
- Một dãy ô tường liên thông với nhau từ bên trái dưới đến bên trên phải của lối.



Hướng tiếp cận:

- Xem ô có ký tự @ là ô bị chặn
- Một ô có thể đi sang 8 hướng khác nhau (những ô chung cạnh hoặc chung góc).
- Tìm đường đi có ít ký tự nhất từ phía trái dưới đến phải trên của lối.

Để giải quyết, ta sẽ sử dụng thuật toán BFS 0-1. Ta cũng cần xét thêm các trường hợp khác như các ô $(1,1)$ hay (r,c) có thể biến thành ô tường, $n = 1, m = 1, \dots$

2.9.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;

int dx[] = { 0, 0, -1, 1, -1, -1, 1, 1 };
int dy[] = { -1, 1, 0, 0, 1, -1, 1, -1 };

int n, m;
vector<vector<char>> a;
vector<vector<int>> d;

bool inGrid(int r, int c) {
    return (r >= 1 && r <= n && c >= 1 && c <= m) ? true : false;
}

deque<pair<int, int>> Q;

void solve() {
    cin >> n >> m;
```

```

if (n == 0 && m == 0)
    exit(0);
a.assign(n + 1, vector<char>(m + 1));
d.assign(n + 1, vector<int>(m + 1));
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
        cin >> a[i][j], d[i][j] = n * m;

if (min(n, m) >= 2) {
    int ans = n * m;
    if (a[1][1] == '#' || a[n][m] == '#') ans = 0;
    if (a[1][1] == '.' || a[n][m] == '.') ans = min(ans, 1);
    Q.clear();
    for (int i = 2; i <= n; i++)
        if (a[i][1] == '#')
            Q.push_front({ i, 1 }), d[i][1] = 0;
        else if (a[i][1] == '.')
            Q.push_back({ i, 1 }), d[i][1] = 1;
    for (int i = 2; i < m; i++)
        if (a[n][i] == '#')
            Q.push_front({ n, i }), d[n][i] = 0;
        else if (a[n][i] == '.')
            Q.push_back({ n, i }), d[n][i] = 1;

    while (!Q.empty()) {
        auto [u, v] = Q.front();
        Q.pop_front();

        if ((u == 1 && v > 1) || (u < n && v == m)) {
            ans = min(ans, d[u][v]);
            break;
        }
        for (int i = 0; i < 8; i++) {
            int r = u + dx[i], c = v + dy[i];
            if (!inGrid(r, c) || a[r][c] == '@') continue;
            int w = a[r][c] == '#' ? 0 : 1;
            if (d[r][c] > d[u][v] + w) {
                d[r][c] = d[u][v] + w;
                if (w) Q.push_back({ r, c });
                else Q.push_front({ r, c });
            }
        }
    }
}

```

```

        }
    }
}

if (ans == n * m) ans = -1;
cout << ans << "\n";
}

else {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (a[i][j] == '#') {
                cout << 0 << "\n";
                return;
            }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (a[i][j] != '@') {
                cout << 1 << "\n";
                return;
            }
    cout << -1 << "\n";
}
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    while (true)
        solve();
}

```

3. Bài tập tự luyện

3.1. Bài tập có hướng dẫn

3.1.1. Fire Again (CF35C)

Link chứa đề: <https://codeforces.com/contest/35/problem/C>

Tóm tắt đề

Có một khu rừng với các cây được trồng theo dạng lưới hình chữ nhật gồm N hàng và M cột. Một đám cháy xảy ra tại K vị trí, sau mỗi giây đám cháy sẽ lan sang những cây kè nó mà chưa bị cháy. Tìm cây bị cháy sau cùng, và thời gian mà nó cháy.

Cách giải

Ta thực hiện BFS trên lưới như bình thường, chỉ cần đẩy vào hàng đợi K vị trí bị cháy ngay tại thời điểm bắt đầu chạy BFS. ĐPT $O(N \times M)$

3.1.2. You and Me (PUCMM223)

Link chứa đề: <https://www.spoj.com/problems/PUCMM223/>

Tóm tắt đề

Trò chơi hai người “You and me” trên bảng kích thước $M \times N$ ($M, N \leq 20$) : Mỗi người có một quân cờ, kí tự ‘a’ và kí tự ‘b’. Trong mỗi lượt, hai người đồng thời di chuyển quân cờ của mình (có thể đứng im tại chỗ). Hai quân cờ này không được đi băng qua nhau hay đứng chung một ô, và không được đi vào ô bị cản. Mục tiêu của trò chơi là đưa quân ‘b’ về vị trí của quân ‘a’ lúc đầu và đưa quân ‘a’ về vị trí của quân ‘b’ lúc đầu. Tìm thời gian ngắn nhất để hoàn thành trò chơi.

Cách giải

Coi mỗi một trạng thái gồm vị trí hiện của ‘a’ (x, y) và ‘b’ (u, v) như là một đỉnh của đồ thị (x, y, u, v) . Mỗi khi hai quân cờ này di chuyển tới (x', y') và (u', v') , ta có được trạng thái mới là (x', y', u', v') và như vậy coi như là đã đi tới một đỉnh khác. Việc di chuyển của hai quân cờ được xem như là một cạnh của đồ thị. Cần tìm đường đi ngắn nhất từ đỉnh (x_a, y_a, x_b, y_b) tới đỉnh (x_b, y_b, x_a, y_a) (bằng BFS). ĐPT $O(m^2 \times n^2)$

3.1.3. Meeting For Part (DCEPC706)

Link chứa đề: <https://www.spoj.com/problems/DCEPC706/>

Tóm tắt đề

Cho vị trí nhà ở của ba người bạn trong bản đồ thành phố, được miêu tả dưới dạng lưới hình chữ nhật $M \times N$ và bao gồm những ô bị chặn. Tìm thời gian ngắn nhất để ba bạn cùng gặp nhau tại một ô, nằm trong hoặc nằm ngoài lưới HCN. Dữ liệu đảm bảo có cách để ba bạn này gặp được nhau tại một ô bất kì.

Cách giải

Vì ba người bạn có thể gặp nhau bên ngoài phần bản đồ thành phố được cho trong input, nên ta cần phải ‘mở rộng’ bản đồ này ra : Tạo thêm M hàng bên trên, M hàng bên dưới, N cột bên trái, N cột bên phải. Độ phức tạp $O(T \times M \times N)$ nếu dùng BFS trên lưới.

3.1.4. Famous Grid (SPIRALGR)

Link chứa đề: <https://www.spoj.com/problems/SPIRALGR/>

Tóm tắt đề

Cho vòng xoáy số nguyên tố (xem ảnh + giải thích trong đề). Những ô chứa số nguyên tố là ô bị chặn. Trả lời T truy vấn tìm đường đi ngắn nhất từ ô chứa hợp số x tới ô chứa hợp số y .

Cách giải

$1 \leq x, y \leq 10^4$ nên chỉ cần dựng trước một bảng kích thước 100×100 là đủ. Vì có khả năng đường đi sẽ ra bên ngoài lưới nên cần có bảng kích thước 101×101 . Sàng nguyên tố (với độ phức tạp $X \times \sqrt{X}$ vẫn được) để đánh dấu những ô bị chặn. Với mỗi truy vấn, tìm đường đi ngắn nhất từ ô chứa số x tới ô chứa số y bằng thuật toán BFS trên lưới.

3.1.4. Fillomino 2 (CF 1517C)

Link chứa đề: <https://codeforces.com/contest/1517/problem/C>

Tóm tắt đề

Cho một hoán vị từ gồm n số từ 1 đến n . Ta cần điền các số đó theo đúng thứ tự trên đường chéo chính của bảng từ góc trái trên đến góc phải dưới $(1, 1) \rightarrow (n, n)$. Yêu cầu điền các số xuống góc dưới phải của bảng sao cho mỗi miền liên thông số X có đúng X ô. Và mỗi ô thuộc một miền đều liên thông với nhau

Giải thuật

Có thể nhận thấy đây là một bài toán sử dụng thuật toán loang trên bảng. Nhưng chúng ta không thể loang theo một phương pháp thông thường.

- Nhận xét rằng tất cả các ô ở nửa dưới bên trái bảng đều phải được điền vào, không bỏ sót ô nào và các ô cùng một số phải liên thông với nhau.
- Vậy thứ tự điền của chúng ta ưu tiên lần lượt là (Trái, Dưới, Phải, Trên).
- Nếu chúng ta đang ở ô (i, j) , nếu có thể di chuyển đến ô $(i, j - 1)$ thì sẽ đi, nếu không tiếp tục xét ô $(i + 1, j)$

Cài đặt

Link source code : [Submission #114181407 - Codeforces](#).

3.2. Bài tập khác

3.2.1. Minimum Knight moves (NAKANJ)

Link chứa đề: <https://www.spoj.com/problems/NAKANJ/>

3.2.2. Maze Escape (11931)

Link chứa đề:
https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=3082

3.2.3. SC (SC)

Link chứa đề: <http://lequydon.ntucoder.net/Problem/Details/4691>

3.2.4. Đường đi (OLP_CT20_ROUTE)

Link chứa đề: https://oj.vnoi.info/problem/olp_ct20_route

3.2.5. Shortest distance between two cells in a matrix or grid

Link chứa đề: <https://www.geeksforgeeks.org/shortest-distance-two-cells-matrix-grid/>

3.2.6. Shortest Path in a Binary Weight Graph

Link chứa đề: <https://www.geeksforgeeks.org/0-1-bfs-shortest-path-binary-graph/>

Kết luận

Chuyên đề này mô tả bốn thuật toán để xử lý các bài toán về Tìm đường đi ngắn nhất trên đồ thị lưới:

- Thuật toán BFS trên đồ thị lưới
- Thuật toán Dijkstra trên đồ thị lưới
- Thuật toán BFS 0-1 trên đồ thị lưới
- Dial's Algorithm

Ngoài hai phương pháp quen thuộc BFS và Dijkstra được áp dụng phổ biến thì thuật toán BFS 0-1 áp dụng để tối ưu trong các trường hợp riêng của BFS, thuật toán Dial là trường hợp riêng để giải quyết những bài toán áp dụng Dijkstra với cấu trúc dữ liệu Heap không hiệu quả.

Dựa trên các phương pháp mô tả, có thể triển khai các kế hoạch giảng dạy khác nhau phù hợp với đối tượng học sinh.

Tài liệu tham khảo

1. Tài liệu tập huấn phát triển chuyên môn giáo viên trường THPT chuyên, môn Tin học, 2012
2. Giải thuật và lập trình, Lê Minh Hoàng, Đại học Sư phạm HN
3. Đề thi Duyên Hải Đồng Bằng Bắc Bộ năm 2020, 2021
4. Đề thi 2020 ICPC Vietnam National
5. Đề thi Olympic Sinh viên năm 2020
6. Các bài tập của thầy Đỗ Đức Đông – Đại học Công Nghệ - ĐHQG HN
7. Các bài tập của thầy Nguyễn Đức Nghĩa – Đại học Bách Khoa HN
8. Một số trang web
 - a. <https://codeforces.com>
 - b. <https://www.spoj.com>
 - c. <https://www.geeksforgeeks.org>
 - d. <https://onlinejudge.org>
 - e. <https://oj.vnoi.info>
 - f. <https://lqdoj.edu.vn>
 - g. <http://lequydon.ntucoder.net>