1 BuildingRoads

Cho N thành phố được đánh số từ 1 đến N, và M con đường đã xây giữa một số cặp thành phố. Mỗi con đường nối hai thành phố a và b.

Mục tiêu là thêm vào một số tối thiểu các con đường sao cho mọi thành phố đều nằm trong cùng một thành phần liên thông.

1. Cấu trúc DSU

Mỗi thành phố ban đầu là một tập riêng biệt.

- Mảng `ds[i]` lưu thông tin về gốc của tập chứa thành phố i.
- Hàm `find(u)` trả về gốc của tập chứa u, đồng thời áp dụng *path compression*:

$$\text{find}(u) = \begin{cases} u & \text{nếu } ds[u] < 0 \\ \text{find}(ds[u]) & \text{ngược lại} \end{cases}$$

- Hàm `merge(u, v)` hợp nhất hai tập nếu u và v thuộc hai tập khác nhau:

$$\text{merge}(u, v) = \begin{cases} \text{false} & \text{nếu find}(u) = \text{find}(v) \\ \text{gộp hai tập và trả về true} & \text{ngược lại} \end{cases}$$

2. Thuật toán chính

1. Khởi tạo: `ds[i] = -1` với mọi i từ 1 đến N.
2. Với mỗi cạnh (a, b) đầu vào, gọi `merge(a, b)` để nối các thành phố đã liên thông.
3. Duyệt từ $i = 1$ đến $N - 1$:
   - Nếu `merge(i, i+1)` thành công, nghĩa là i và $i + 1$ chưa liên thông, thì thêm cạnh $(i, i + 1)$ vào danh sách kết quả.
4. In ra số đường thêm vào và danh sách các đường đó.

3. Độ phức tạp

Với kỹ thuật *path compression* và *union by size*, mỗi phép `find` hay `merge` có độ phức tạp gần $\mathcal{O}(1)$ (chính xác là $\mathcal{O}(\alpha(N))$ với α là hàm nghịch đảo Ackermann).

Tổng độ phức tạp:

$$\mathcal{O}(M \cdot \alpha(N) + N \cdot \alpha(N)) \approx \mathcal{O}(N)$$

2 CountingRooms

Mô tả bài toán

Cho một mê cung dưới dạng lưới $N \times M$ với mỗi ô là:

- `.` — ô trống có thể đi vào.
- `#` — tường không thể đi qua.

Hai ô trống được xem là **liên thông** nếu chúng kề nhau theo hướng lên, xuống, trái, hoặc phải.

Yêu cầu: Đếm số vùng liên thông các ô trống — gọi là *số phòng*.

Ý tưởng thuật toán

1. Duyệt toàn bộ ma trận.
2. Với mỗi ô chưa được thăm và là ô trống `'.'`, thực hiện thuật toán DFS để đánh dấu tất cả các ô trong vùng liên thông.
3. Mỗi lần gọi DFS tương ứng với một phòng mới.

Hàm DFS

Giả sử đang ở ô (x, y), đánh dấu là đã thăm:

$$\text{visited}[x][y] \leftarrow \text{true}$$

Sau đó, thử di chuyển sang 4 hướng:

$$(x+1, y),\ (x-1, y),\ (x, y+1),\ (x, y-1)$$

Nếu ô mới hợp lệ, không phải tường, chưa được thăm, thì tiếp tục gọi đệ quy.

Độ phức tạp thời gian

Mỗi ô được thăm nhiều nhất 1 lần, nên tổng thời gian là:

$$\mathcal{O}(N \cdot M)$$

3 Labyrinth

Bài toán

Cho mê cung dạng lưới kích thước $N \times M$. Mỗi ô có thể là:

- Tường (#): không đi được
- Đường đi (.)

- Điểm bắt đầu: A
- Điểm kết thúc: B

Yêu cầu: Tìm đường đi ngắn nhất từ A đến B, hoặc trả lời rằng không thể.

Mô hình hoá bài toán

Mỗi ô trong lưới là một đỉnh trong đồ thị. Các đỉnh kề nhau nếu chúng là ô trống và liền kề theo 4 hướng:

$$\text{Hướng dịch chuyển: } \begin{cases} \text{Xuống (D)} : (x+1, y) \\ \text{Lên (U)} : (x-1, y) \\ \text{Phải (R)} : (x, y+1) \\ \text{Trái (L)} : (x, y-1) \end{cases}$$

Giải thuật BFS

1. Khởi tạo hàng đợi Q, đánh dấu ô bắt đầu A là đã thăm.
2. Trong khi Q chưa rỗng:
   (a) Lấy phần tử đầu hàng đợi: (x, y)
   (b) Duyệt 4 hướng để tìm ô kề chưa thăm và không phải tường
   (c) Đánh dấu đã thăm, lưu hướng đi, cập nhật khoảng cách:
   $$\text{dist}[x'][y'] = \text{dist}[x][y] + 1$$
   (d) Thêm ô (x', y') vào hàng đợi
3. Khi kết thúc, nếu ô B chưa được thăm, trả lời `NO`
4. Ngược lại, in ra đường đi bằng cách lần ngược từ B về A theo mảng hướng đi `par`

Độ phức tạp

- Thời gian: $O(N \times M)$, vì mỗi ô được duyệt tối đa 1 lần
- Không gian: $O(N \times M)$ cho các mảng: `vis`, `dist`, `par`

4 MessageRoute

Đề bài

Cho một đồ thị vô hướng gồm N đỉnh và M cạnh, đánh số từ 1 đến N. Hãy tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh N. Nếu không tồn tại, in ra `IMPOSSIBLE`.

Input:

- Dòng đầu tiên chứa hai số nguyên N, M
- M dòng tiếp theo, mỗi dòng chứa hai số nguyên a, b biểu diễn cạnh nối giữa đỉnh a và đỉnh b

Output:

- Nếu không tồn tại đường đi từ 1 đến N, in `IMPOSSIBLE`
- Ngược lại, in số bước và các đỉnh trên đường đi ngắn nhất

Ý tưởng và giải thuật

Sử dụng thuật toán **BFS (Tìm kiếm theo chiều rộng)** để tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh N.

4.0.1 Mô hình hóa đồ thị

Đồ thị được biểu diễn dưới dạng danh sách kề:

$$G[u] = \text{danh sách các đỉnh kề với } u$$

4.0.2 Biến và cấu trúc dữ liệu sử dụng

- `vis[u]`: đánh dấu đỉnh u đã được thăm
- `dist[u]`: độ dài đường đi ngắn nhất từ đỉnh 1 đến u
- `p[u]`: đỉnh trước đó trên đường đi đến u

4.0.3 Thuật toán BFS

1. Khởi tạo hàng đợi Q, đưa đỉnh 1 vào Q, đánh dấu đã thăm
2. Trong khi Q chưa rỗng:
   (a) Lấy $u = Q.\text{front}()$, xóa khỏi hàng đợi
   (b) Duyệt tất cả đỉnh v kề với u
   (c) Nếu v chưa thăm:
      - Cập nhật $\text{dist}[v] = \text{dist}[u] + 1$
      - Cập nhật $p[v] = u$
      - Đánh dấu đã thăm và đưa vào hàng đợi
3. Sau khi BFS hoàn tất:
   - Nếu N chưa được thăm $\Rightarrow$ `IMPOSSIBLE`
   - Ngược lại, dùng mảng `p` để truy vết đường đi từ N về 1

4.1 Độ phức tạp

- Thời gian: $\mathcal{O}(N + M)$
- Không gian: $\mathcal{O}(N + M)$