**Comparisons**

**I. Mazes: Compare search behavior (no animation)**

**I.1. Mục tiêu:**

Bài thực hành này nhằm so sánh hành vi tìm kiếm của các thuật toán khác nhau trong việc giải quyết bài toán tìm đường trong mê cung (maze), bao gồm:

- **BFS** (Breadth-First Search) - Tìm kiếm theo chiều rộng

- **DFS** (Depth-First Search) - Tìm kiếm theo chiều sâu

- **GBFS** (Greedy Best-First Search) - Tìm kiếm tham lam

- **A\***(A-Star Search) - Thuật toán A\*

- **IDS** (Iterative Deepening Search) - Tìm kiếm lặp tăng dần độ sâu

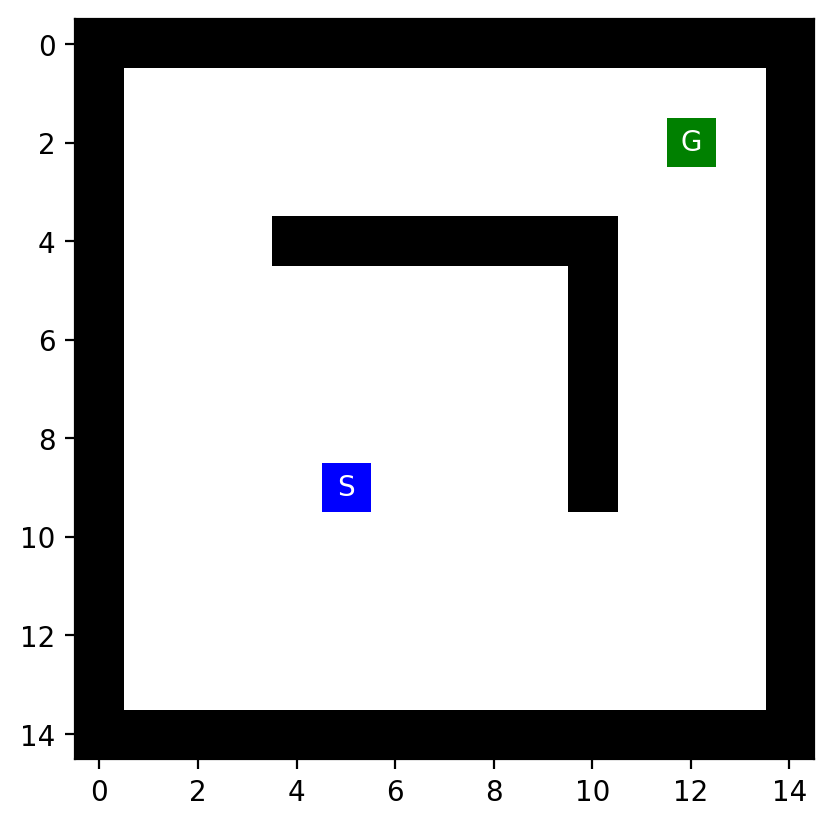
Thực nghiệm thực hiện trên mê cung L\_maze.txt với:

-Điểm xuất phát (S): Tọa độ (6, 9)

-Điểm đích (G): Tọa độ (13, 2)

-Các vật cản (tường) màu đen

-Không gian di chuyển màu xám/trắng



Sử dụng **Manhattan Distance** làm hàm **heuristic**: **h(n) = |x1 - x2| + |y1 - y2|**

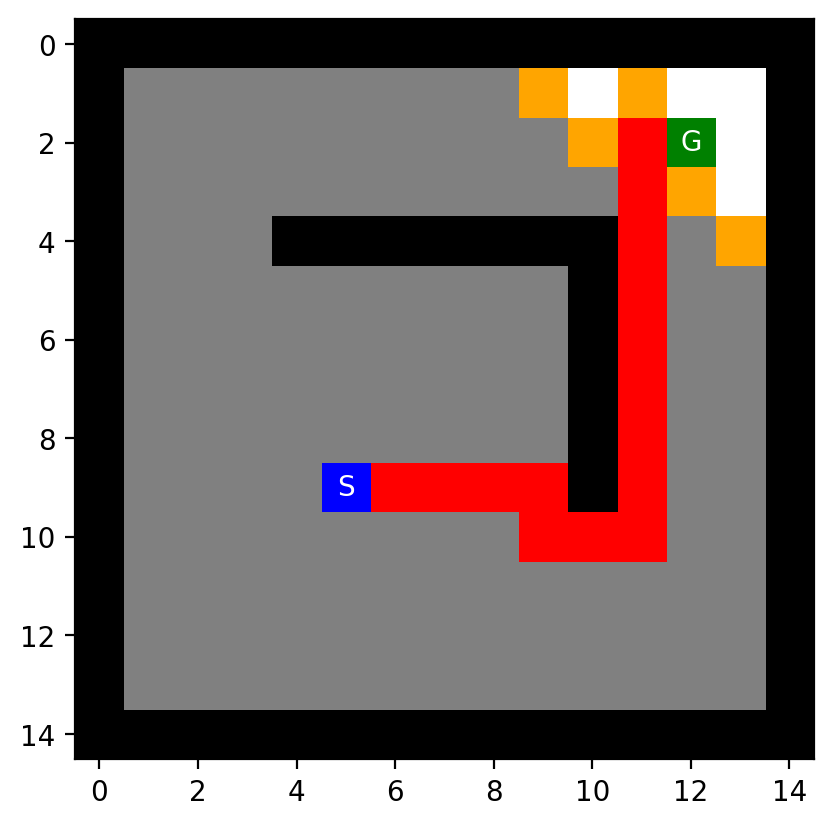
**I.2 Phân tích**

**I.2.1 BFS (Breadth-First Search)**

- Khám phá tất cả các node ở độ sâu k trước khi chuyển sang độ sâu k+1

- Sử dụng cấu trúc Queue (FIFO - First In First Out)

- Đảm bảo tìm được đường đi ngắn nhất



**Đặc điểm:**

**+Ưu điểm:**

  - Đảm bảo tìm được đường đi ngắn nhất

  - Hoàn chỉnh (luôn tìm được nghiệm nếu có)

+**Nhược điểm:**

  - Khám phá nhiều ô không cần thiết (phần màu xám)

  - Tiêu tốn bộ nhớ cao với không gian rộng

**Hành vi tìm kiếm:**

- Mở rộng đồng đều theo tất cả hướng từ điểm xuất phát

- Khám phá các vùng xung quanh một cách có hệ thống

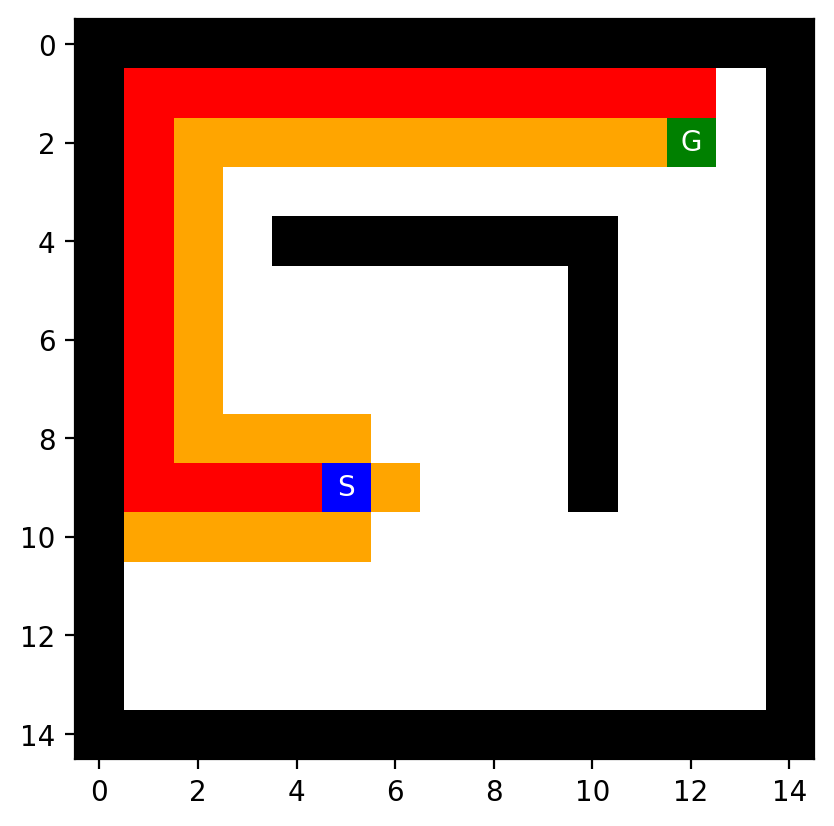
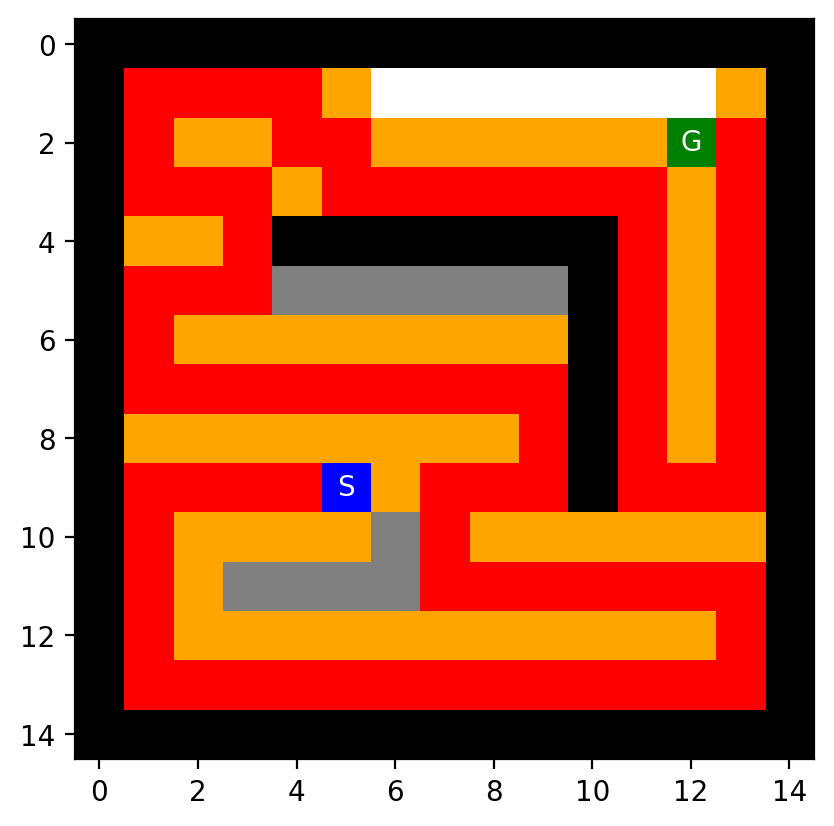
- Màu xám cho thấy phạm vi khám phá rất rộng

**I.2.2** **DFS (Depth-First Search)**

- Đi sâu vào một nhánh cho đến khi gặp bế tắc hoặc đích

- Sử dụng cấu trúc Stack (LIFO - Last In First Out)

- Không đảm bảo tìm được đường đi ngắn nhất



*ts****.****set\_order("NESW") tree\_search****.****set\_order("SENW")*

**Đặc điểm:**

+**Ưu điểm:**

  - Thời gian chạy nhanh nhất

  - Tiết kiệm bộ nhớ hơn BFS trong một số trường hợp

+**Nhược điểm:**

  - Đường đi tìm được rất dài (80 bước so với 16 bước của BFS)

  - Phụ thuộc nhiều vào thứ tự khám phá các hướng

  - Có thể đi vào đường vòng không hiệu quả

**Hành vi tìm kiếm:**

- Đi theo một hướng sâu trước khi quay lui

- Tạo ra đường đi quanh co, không hiệu quả

- Đường màu đỏ cho thấy nhiều đoạn đi vòng không cần thiết

**Lưu ý về độ phức tạp không gian:**

- Implementation này giữ toàn bộ cây tìm kiếm trong bộ nhớ (cấu trúc "reached")

- Độ phức tạp không gian: O(b^m) - tệ hơn BFS!

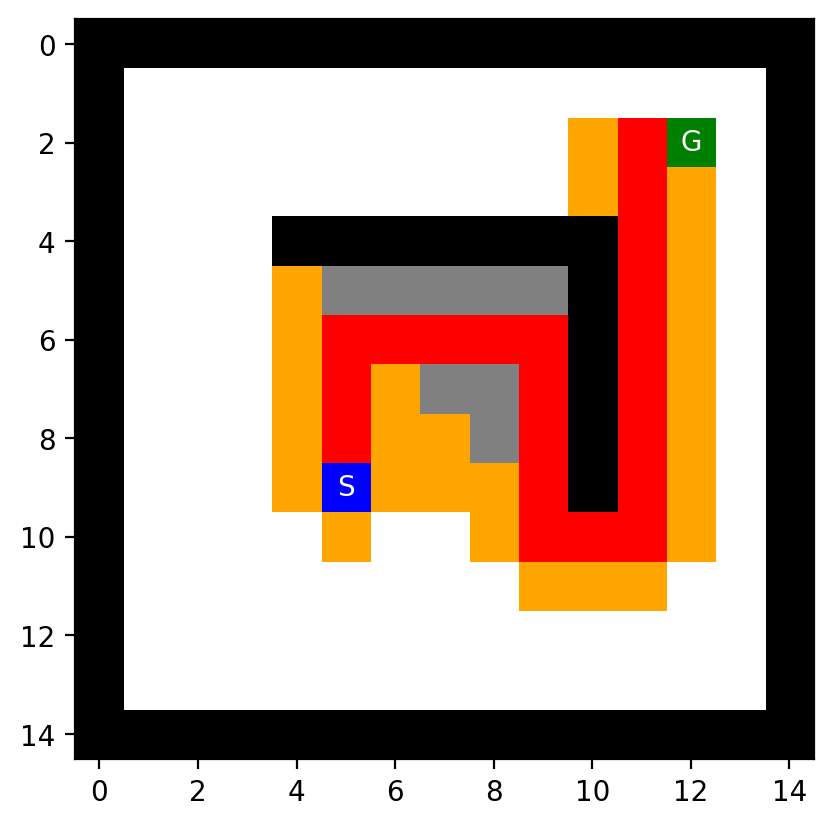
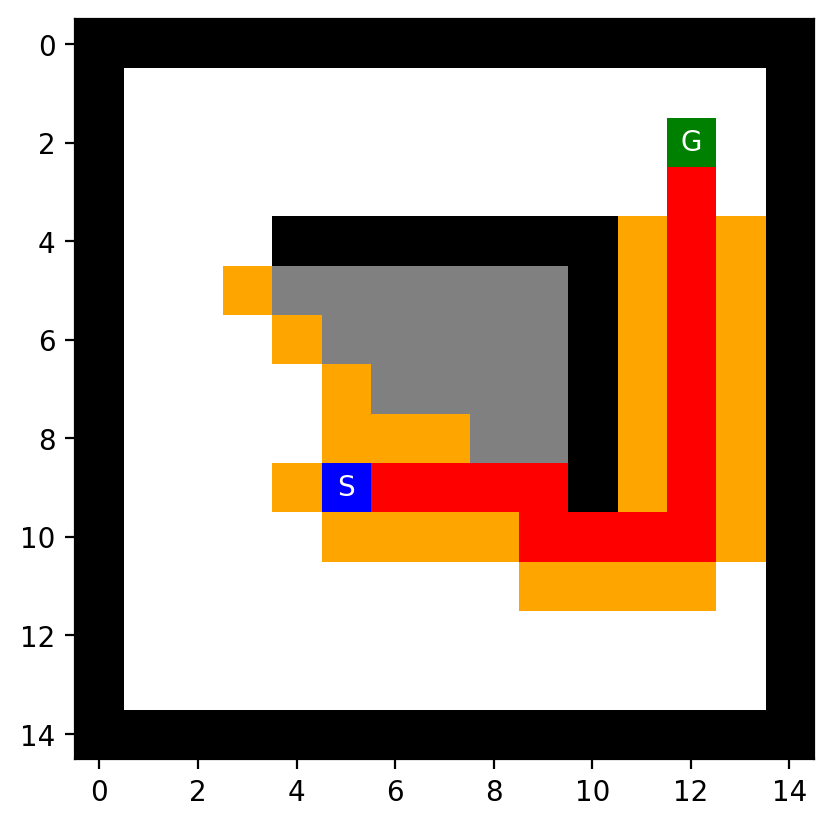
- DFS không có cấu trúc "reached" có độ phức tạp O(bm) nhưng có thể bị vòng lặp vô hạn

**I.2.3** **GBFS (Greedy Best-First Search)**

- Chọn node có giá trị heuristic h(n) nhỏ nhất (gần đích nhất)

- Tham lam - chỉ xét khoảng cách đến đích, không quan tâm đường đã đi

- Không đảm bảo tìm được đường đi tối ưu



*tree\_search****.****set\_order("NESW") tree\_search****.****set\_order("WSEN")*

**Đặc điểm:**

+**Ưu điểm:**

  - Khám phá ít ô nhất (61 ô)

  - Tập trung vào hướng đích

  - Hiệu quả cao trong mê cung đơn giản

+**Nhược điểm:**

  - Không đảm bảo tối ưu trong tất cả trường hợp

  - Có thể bị "lừa" bởi heuristic không chính xác

**Hành vi tìm kiếm:**

- Tập trung khám phá theo hướng đích

- Vùng màu cam rất hẹp, cho thấy tính định hướng cao

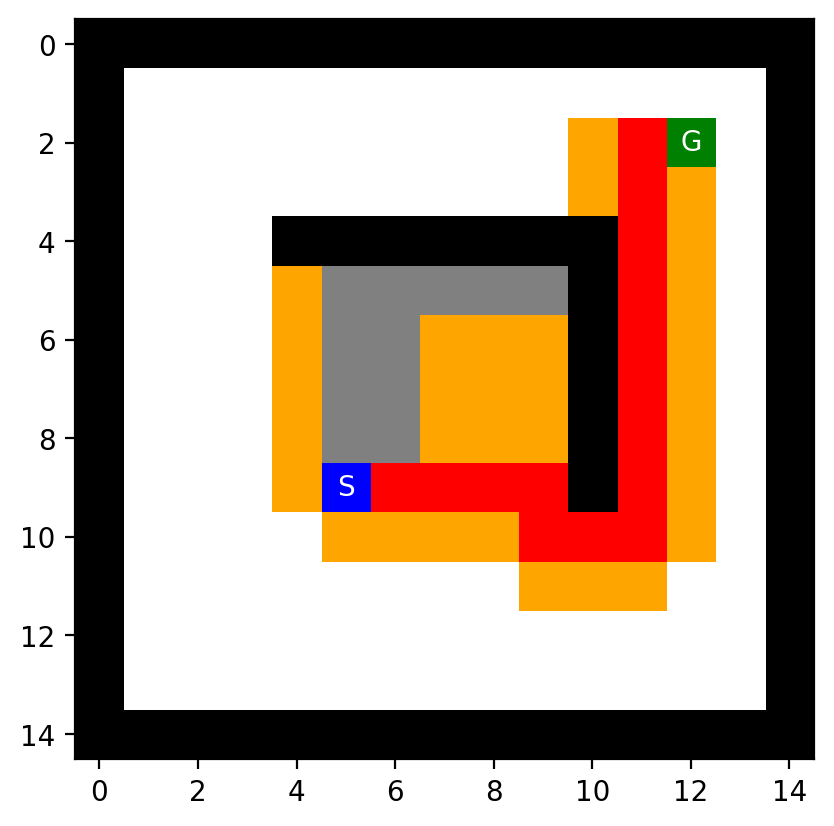
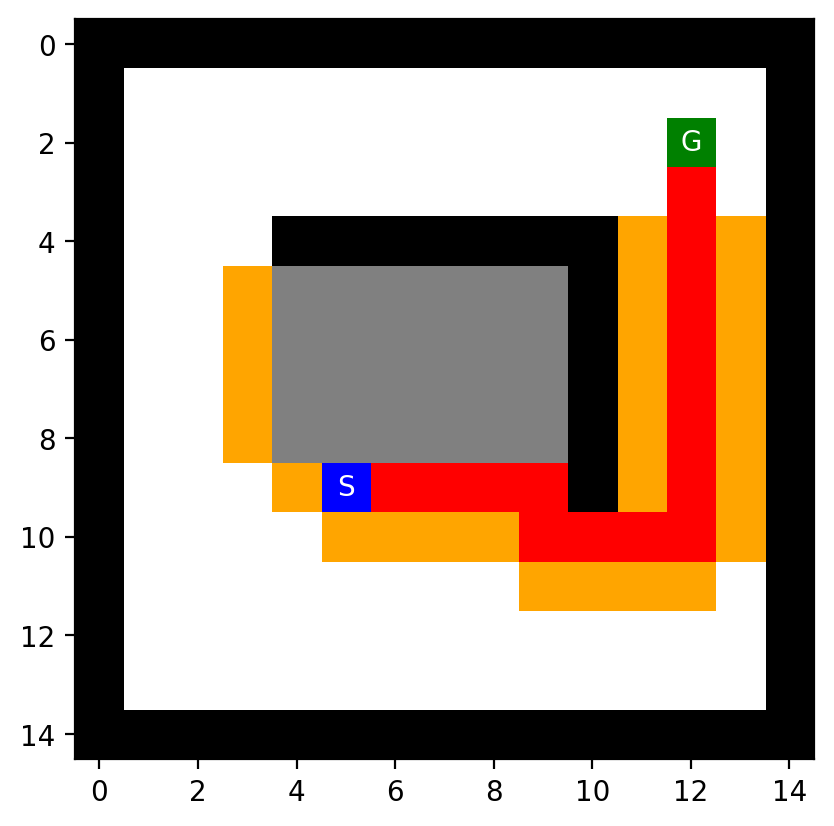
- Đường đi trực tiếp, ít quanh co

**I.2.4 A\* (A-Star Search)**

- Kết hợp chi phí đã đi g(n) và ước lượng chi phí còn lại h(n)

- Đánh giá node theo f(n) = g(n) + h(n)

- Đảm bảo tối ưu nếu heuristic admissible (h(n) ≤ chi phí thực tế)



*tree\_search****.****set\_order("NESW") tree\_search****.****set\_order("WSEN")*

**Đặc điểm:**

+**Ưu điểm:**

  - Đảm bảo tìm được đường đi ngắn nhất

  - Hiệu quả hơn BFS (chỉ 66 ô so với 151 ô)

  - Cân bằng giữa tối ưu và hiệu suất

+**Nhược điểm:\*\***

  - Phức tạp hơn BFS và DFS

  - Phụ thuộc vào chất lượng heuristic

**Hành vi tìm kiếm:**

- Cân bằng giữa chi phí đã đi và ước lượng còn lại

- Khám phá có định hướng nhưng vẫn đảm bảo tối ưu

- Vùng khám phá hẹp hơn BFS nhưng rộng hơn GBFS một chút

**I.3 Hiệu suất**

**\*Thuật toán tốt nhất: A\***

- Tìm được đường đi tối ưu (16 bước)

- Hiệu quả cao (chỉ 66 ô khám phá)

- Thời gian nhanh (1.64 ms)

- Đảm bảo tính hoàn chỉnh và tối ưu

**\*Nhanh nhất: DFS**

- Thời gian thực thi ngắn nhất 1.18 ms

- Nhưng đường đi rất dài (80 bước - gấp 5 lần tối ưu!)

- Không phù hợp khi cần đường đi ngắn

**\*Hiệu quả nhất: GBFS**

- Khám phá ít ô nhất (61 ô)

- May mắn tìm được đường tối ưu trong trường hợp, nhưng không đảm bảo tối ưu trong mọi trường hợp

**\*Chuẩn xác nhất: BFS**

- Luôn tìm được đường ngắn nhất

- Nhưng tốn kém về không gian (151 ô)

- Phù hợp khi không có heuristic tốt

**I.4 Ảnh hưởng của thứ tự khám phá**

**I.4.1. Thứ tự NESW vs SENW vs WSEN vs Random**

Thử nghiệm cho thấy thứ tự khám phá các hướng (North, East, South, West) có ảnh hưởng lớn đến:

\*Với DFS:

- Thứ tự khác nhau → Đường đi hoàn toàn khác nhau

- Độ dài đường đi có thể thay đổi đáng kể

- Cần chạy nhiều lần với random và chọn kết quả tốt nhất

\*Với BFS và A\*:

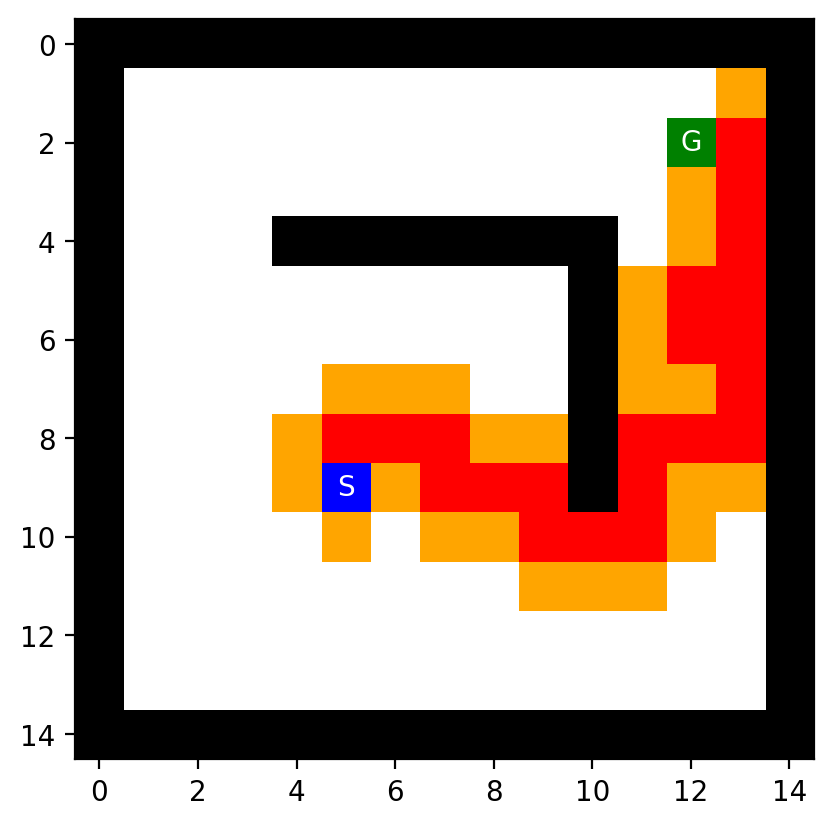
- Thứ tự ảnh hưởng ít hơn

- Vẫn tìm được đường tối ưu

- Có thể ảnh hưởng đến thời gian thực thi

**I.4.2. Random DFS - Chạy nhiều lần**

Notebook có thử nghiệm chạy DFS random 100 lần và chọn kết quả tốt nhất:



- Có thể tìm được đường đi tốt hơn

- Tốn thời gian nhưng không đảm bảo tối ưu

- IDS (Iterative Deepening Search) là giải pháp tốt hơn

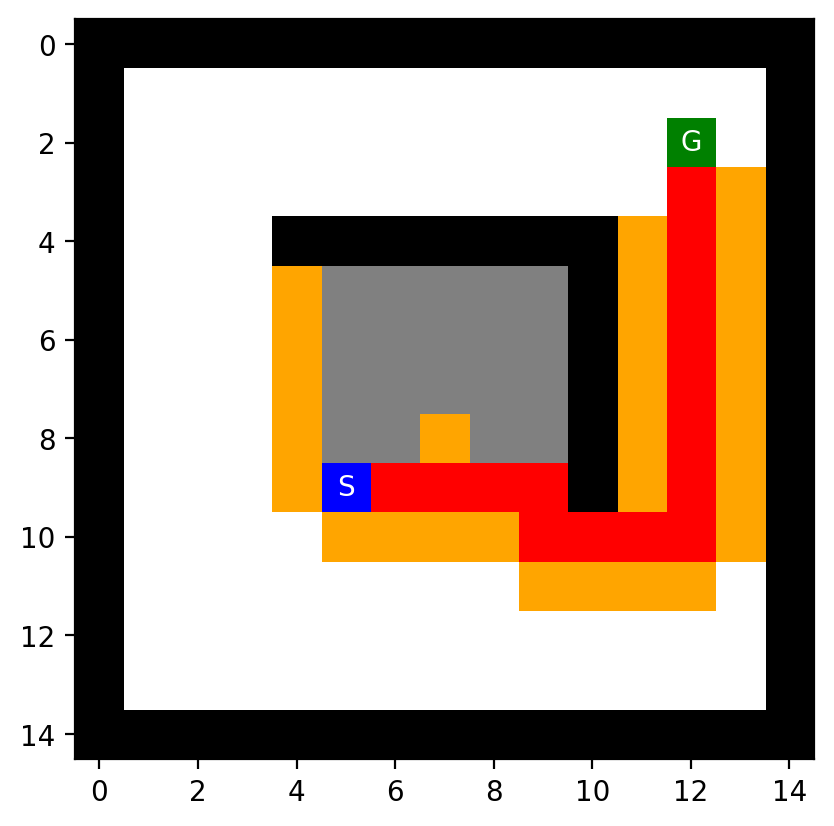
**I.5. Biến thể của A\*: Weight A\***

**I.5.1. Công thức**

f(n) = g(n) + W × h(n)

**I.5.2. Ảnh hưởng của trọng số W**

**\*W > 1:** Xu hướng về GBFS

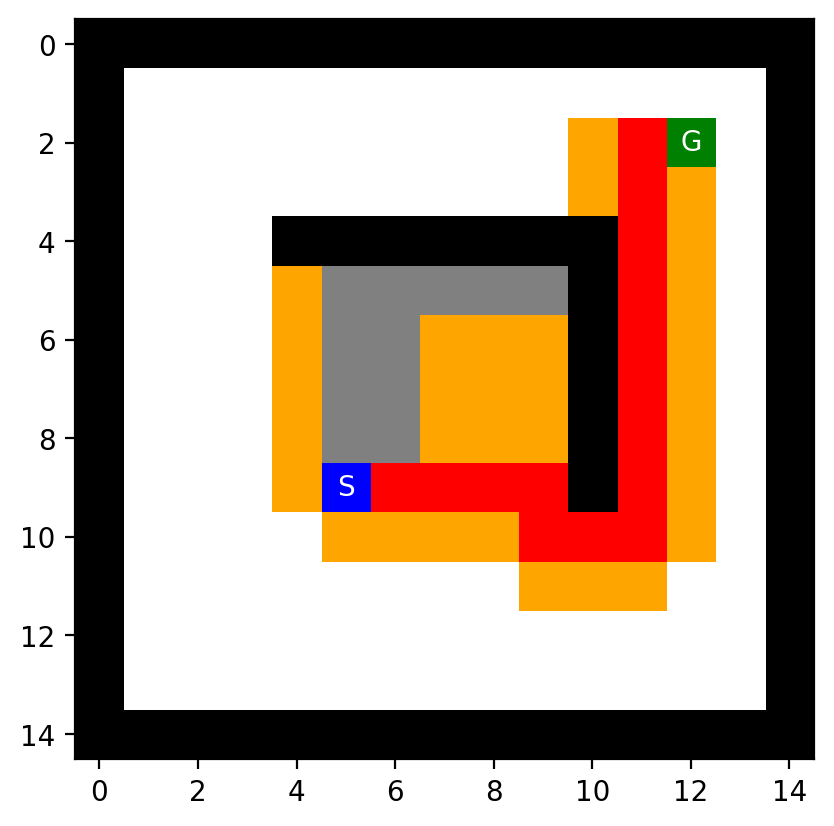


- W càng lớn → càng tham lam

- Không đảm bảo tối ưu

- Có thể nhanh hơn trong một số trường hợp

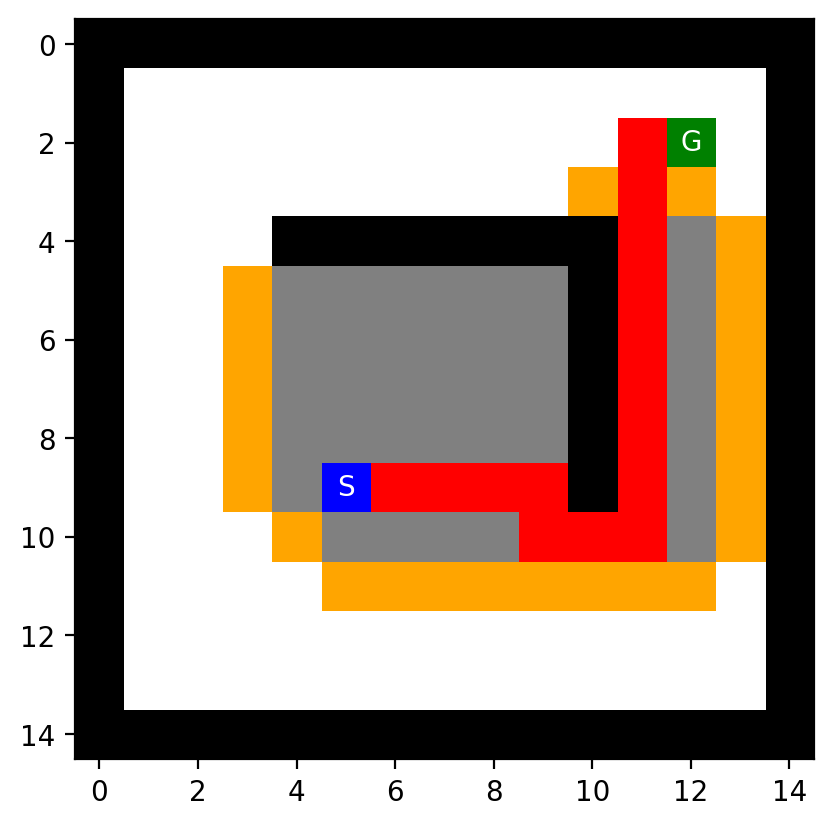
**\*W = 1:** A\* chuẩn



- Cân bằng tối ưu

- Đảm bảo tìm đường ngắn nhất

**\*W < 1:** Xu hướng về BFS/UCS



- W càng nhỏ → càng giống BFS

- Đảm bảo tối ưu

- Có thể chậm hơn

**I.6. Kết luận**

1. **A\* là thuật toán cân bằng nhất** cho bài toán tìm đường trong mê cung:

   - Đảm bảo tối ưu

   - Hiệu quả cao (ít khám phá)

   - Thời gian chấp nhận được

2.**\*BFS đảm bảo tối ưu** nhưng kém hiệu quả về không gian

3. **DFS nhanh nhưng cho kết quả kém** - không nên dùng khi cần đường đi ngắn

4. **GBFS hiệu quả nhưng không đảm bảo** - phù hợp khi cần nhanh và chấp nhận không tối ưu

5. **Heuristic tốt là chìa khóa**- Manhattan distance hoạt động tốt cho mê cung 2D