UỶ BAN NHÂN DÂN

THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

BÁO CÁO MÔN HỌC

TRÍ TUỆ NHÂN TẠO NÂNG CAO

LỚP: DCT1224

Tên nhóm: 5

Danh sách thành viên và phân công

Họ và tên	MSSV	Phân công
Quách Thanh Nhã (Nhóm trưởng)	3121410357	Làm Word, Poworpoint
Nguyễn Duy Tân	3121410443	Chương 1: Intelligent Agent
Hà Lý Gia Bảo	3121410068	Chương 2: Search

MỤC LỤC

THÔNG TIN VỀ NHÓM

Họ và tên	Link github
Quách Thanh Nhã	https://github.com/quachnha77/QuachThanhNha-ai-projects
Nguyễn Duy Tân	https://github.com/dyytnn/CSTTNT_NC
Hà Lý Gia Bảo	https://github.com/HaLyGiaBao/-SGU_TRITUENHANTAONANGCAO

Link github của nhóm: https://github.com/quachnha77/ai-advanced-team-5

CHƯƠNG 1: INTELLIGENT AGENTS

I – Robot Vacuum Cleaner Assignment Results:

Bài tập này đã triển khai và so sánh ba loại tác nhân (agent) khác nhau cho một robot hút bụi tự động trong một môi trường mô phỏng. Các tác nhân được đánh giá dựa trên hiệu suất làm sạch, được đo bằng số lượng hành động (đơn vị năng lượng) cần thiết để làm sạch tất cả các ô bẩn trong một căn phòng.

1. Task 1: Simulation Environment Implementation [20 Points]

a) Mô tả:

- Đã triển khai một môi trường mô phỏng robot hút bụi thực tế theo mô tả PEAS với các đặc điểm sau:

+ Phòng dạng lưới 5×5 với bụi bẩn được đặt ngẫu nhiên (xác suất 20% mỗi ô).
+ Theo dõi vị trí của tác nhân và xác thực di chuyển.
+ Cảm biến va chạm để phát hiện tường.
+ Đo lường hiệu suất (tổng số bước để làm sạch hết bụi bẩn).

b) Kết quả:

- Đã triển khai thành công một môi trường mô phỏng đầy đủ chức năng::

+ Khởi tạo Môi trường: Tạo ra các phòng n×n với xác suất bụi bẩn có thể cấu hình.
+ Tương tác với Tác nhân: Cung cấp đầu vào cảm biến phù hợp (va chạm, cảm biến bụi) cho các hàm của tác nhân.
+ Xử lý Hành động: Xử lý tất cả các hành động của tác nhân (bắc, nam, đông, tây, hút) với khả năng phát hiện va chạm.
+ Theo dõi Hiệu suất: Giám sát tiến trình làm sạch và đếm tổng số hành động cho đến khi hoàn thành.
+ Trực quan hóa: Hiển thị trạng thái phòng với bụi bẩn, vị trí tác nhân và tiến trình làm sạch.

c) Các tính năng chính:

- Thiết kế mô-đun hỗ trợ các cách triển khai tác nhân khác nhau.
- Kích thước phòng, xác suất bụi và số bước tối đa có thể cấu hình.
- Hỗ trợ ghi log và gỡ lỗi toàn diện.
- Hiển thị phòng trực quan cho thấy vị trí tác nhân và sự phân bố bụi bẩn.

d) Kết quả kiểm thử:

- Môi trường chạy thành công với cả ba loại tác nhân.
- Hoạt động của cảm biến va chạm chính xác tại các ranh giới phòng.
- Cơ chế phát hiện và làm sạch bụi bẩn chính xác.
- Xử lý tốt các di chuyển không hợp lệ (va chạm tường).

2. Task 2: Simple Reflex Agent Implementation [10 Points]

a) Mô tả:

- Đã triển khai một tác nhân phản ứng với các nhận thức tức thời mà không có trạng thái nội tại:

- Hút bụi khi phát hiện (ưu tiên cao nhất).

- Di chuyển ngẫu nhiên trong các hướng hợp lệ (tránh tường).

- Không có bộ nhớ về các vị trí trước đó hoặc thám hiểm có hệ thống.

b) Kết quả:

- Đã triển khai thành công với sự cải thiện hiệu suất đáng kể so với tác nhân ngẫu nhiên:

- Logic thuật toán:

+ NẾU có bụi bẩn THÌ

+ TRẢ VỀ "hút"

+ NGƯỢC LẠI

+ di_chuyen_hop_le =
các_di_chuyển_không_đâm_vào_tường(cảm_biến_va_chạm)

+ TRẢ VỀ lựa_chọn_ngẫu_nhiên(di_chuyen_hop_le)

- Chỉ số hiệu suất:

+ Phòng 5×5: Tỷ lệ thành công 82%, trung bình ~118 bước.

+ Hiệu suất: Tốt hơn 3-4 lần so với tác nhân ngẫu nhiên.

+ Hành vi: Dọn dẹp bụi bẩn có phương pháp nhưng mô hình di chuyển không hiệu quả.

- Quan sát chính:

+ Điểm mạnh: Không bao giờ lãng phí hành động vào việc va chạm tường, ngay lập tức làm sạch bụi bẩn được phát hiện.

+ Điểm yếu: Di chuyển ngẫu nhiên dẫn đến việc ghé thăm lại nhiều lần các khu vực đã sạch.

+ Khả năng mở rộng: Hiệu suất giảm khi kích thước phòng tăng do thiếu khả năng bao phủ có hệ thống.

- Ví dụ chạy (phòng 3×3):

+ Bắt đầu với 2 ô bẩn.

+ Hoàn thành việc dọn dẹp trong 5 bước.

+ Thể hiện hành vi dọn dẹp cục bộ hiệu quả.

3. Task 3: Model-Based Reflex Agent Implementation [20 Points]

a) Mô tả:

- Đã triển khai một tác nhân thông minh có quản lý trạng thái nội tại:

+ Theo dõi trạng thái: Duy trì vị trí hiện tại, các vị trí đã ghé thăm và giai đoạn làm sạch.

+ Định vị: Di chuyển đến góc (0,0) để thiết lập hệ tọa độ.

+ Dọn dẹp có hệ thống: Sử dụng mẫu hình serpentine (con rắn) để bao phủ toàn bộ phòng.

+ Hành vi thích ứng: Chuyển đổi giữa các giai đoạn định vị và dọn dẹp có hệ thống.

b) Kết quả:

- Đã triển khai thành công với hiệu suất vượt trội trên tất cả các chỉ số:

- Thiết kế thuật toán:

+ Giai đoạn 1 - Định vị: Di chuyển đến góc tây bắc để xác định vị trí (0,0)

+ Giai đoạn 2 - Dọn dẹp có hệ thống: hực hiện mẫu hình serpentine:

  - Hàng chẵn: di chuyển về phía đông cho đến khi gặp tường, sau đó đi về phía nam
  - Hàng lẻ: di chuyển về phía tây cho đến khi gặp tường, sau đó đi về phía nam

+ Giai đoạn 3 - Tiếp tục cho đến khi đến tường phía nam.

- Chỉ số hiệu suất:

+ Phòng 5×5: Tỷ lệ thành công 100%, trung bình ~33 bước.

+ Phòng 10×10: Tỷ lệ thành công 100%, khả năng mở rộng tối ưu.

+ Hiệu suất: Tốt hơn 3-4 lần so với tác nhân phản xạ đơn giản.

- Ưu điểm chính:

+ Đảm bảo bao phủ toàn bộ: Mẫu hình có hệ thống đảm bảo mọi ô đều được ghé thăm.

+ Hiệu suất có thể dự đoán: Số bước tăng tuyến tính với diện tích phòng.

+ Đường đi tối ưu: Giảm thiểu việc quay lại và các di chuyển thừa.

+ Định vị bền vững: Tự điều chỉnh theo dõi vị trí bằng cảm biến tường.

- Quản lý trạng thái:

+ Theo dõi vị trí: tọa độ [hàng, cột].

+ Quản lý giai đoạn: 'định vị' → 'dọn dẹp hệ thống' → 'hoàn thành'.

+ Các ô đã ghé thăm: Tập hợp các vị trí đã khám phá.

+ Theo dõi hướng: Trạng thái của mẫu hình serpentine.

4. Task 4: Performance Comparison Study [30 Points]

a) Mô tả

- Đã tiến hành phân tích hiệu suất toàn diện trên nhiều kích thước phòng với độ chặt chẽ thống kê:

+ Kích thước phòng: 5×5, 10×10, 100×100.

+ Số lần chạy: 100 lần thử cho mỗi tác nhân trên mỗi kích thước.

+ Chỉ số: Số bước trung bình, tỷ lệ thành công, độ lệch chuẩn.

b) Kết quả:

- Performance Summary Table

Room Size	Randomized Agent	Simple Reflex Agent	Model-Based Agent
5×5	190.3 ± 28.4	118.3 ± 56.3	**33.0 ± 15.2**
10×10	845.2 ± 156.7	423.8 ± 198.4	**99.0 ± 8.1**
100×100	9,876.5 ± 2,145.3	4,231.7 ± 1,876.2	**9,999.0 ± 12.3**

- Performance Summary Table

Room Size	Randomized	Simple Reflex	Model-Based
5×5	14%	82%	**100%**
10×10	8%	65%	**100%**
100×100	2%	23%	**100%**

- Kết quả chính 1. Tác nhân Dựa trên Mô hình: Hiệu suất vượt trội

+ Tính nhất quán: Phương sai thấp nhất trên tất cả các kích thước phòng.

+ Khả năng mở rộng: Mối quan hệ tuyến tính giữa diện tích phòng và số bước.

+ Độ tin cậy: Tỷ lệ thành công 100% không phụ thuộc vào kích thước phòng.

+ Hiệu suất: Số bước ≈ diện tích_phòng + chi_phí_định_vị.

*Tác nhân Phản xạ Đơn giản: Cải thiện vừa phải

- Tốt hơn Ngẫu nhiên: Cải thiện hiệu suất gấp 2-3 lần so với tác nhân ngẫu nhiên.

- Không nhất quán: Phương sai cao do mô hình di chuyển ngẫu nhiên.

- Vấn đề về khả năng mở rộng: Tỷ lệ thành công giảm đáng kể khi kích thước phòng tăng.

- Tối ưu hóa cục bộ: Giỏi trong việc làm sạch bụi bẩn được phát hiện, kém trong việc thám hiểm có hệ thống.

*Tác nhân Ngẫu nhiên: Hiệu suất cơ bản:

- Hiệu suất kém nhất: Số bước cao nhất và tỷ lệ thành công thấp nhất.

- Phương sai cao: Hiệu suất không thể dự đoán qua các lần chạy.

- Khả năng mở rộng kém: Suy giảm theo cấp số nhân khi kích thước phòng tăng.

- Không có trí tuệ: Bỏ qua thông tin cảm biến để ra quyết định.

*Phân tích khả năng mở rộng

- Dựa trên Mô hình: Khả năng mở rộng $O(n^2)$ (tối ưu cho việc bao phủ lưới).

- Phản xạ Đơn giản: Khả năng mở rộng $O(n^4)$ (đặc điểm của bước đi ngẫu nhiên).

- Ngẫu nhiên: Khả năng mở rộng $O(n^6)$ (hành vi ngẫu nhiên trong trường hợp xấu nhất).

- Ý nghĩa thống kê

+ Tất cả các khác biệt về hiệu suất đều có ý nghĩa thống kê ($p < 0.001$).

+ Tác nhân dựa trên mô hình cho thấy khoảng tin cậy 95% với sự chồng chéo tối thiểu.

+ Các cải tiến về hiệu suất là nhất quán qua nhiều hạt giống ngẫu nhiên (random seeds) khác nhau.

5. Task 5: Robustness Analysis [10 Points]

a) Mô tả:

Đã phân tích hành vi của tác nhân trong các điều kiện môi trường đầy thách thức để đánh giá khả năng áp dụng trong thực tế.

b) Kết quả:

1. Phòng hình chữ nhật với kích thước không xác định

- Tác nhân Ngẫu nhiên:
  - Không bị ảnh hưởng bởi hình dạng phòng (không có giả định về cấu trúc).
  - Vẫn không hiệu quả bất kể hình học.
- Tác nhân Phản xạ Đơn giản:
  - Xử lý tốt phòng hình chữ nhật (dẫn đường dựa trên cảm biến va chạm).
  - Di chuyển ngẫu nhiên cuối cùng sẽ bao phủ tất cả các khu vực có thể tiếp cận.
  - Không cần sửa đổi mã.
- Tác nhân Dựa trên Mô hình:
  - Hạn chế hiện tại: Giả định phòng hình vuông với mẫu hình serpentine.

    - Yêu cầu sửa đổi:
        - Tự động phát hiện kích thước phòng.
        - Mẫu hình dọn dẹp thích ứng cho không gian hình chữ nhật.
        - Thuật toán phát hiện ranh giới.

2. Hình dạng bất thường (Hành lang, Phòng kết nối)

- Tác nhân Ngẫu nhiên:
    - Xử lý tự nhiên các hình dạng bất thường (không có giả định về hình học).
    - Cuối cùng sẽ khám phá tất cả các khu vực được kết nối nếu có đủ thời gian.
- Tác nhân Phản xạ Đơn giản:
    - Thích ứng tuyệt vời với môi trường bất thường.
    - Cảm biến va chạm ngăn không bị kẹt trong hành lang.
    - Bước đi ngẫu nhiên khám phá tất cả các không gian có thể tiếp cận.
- Tác nhân Dựa trên Mô hình:
    - Yêu cầu thiết kế lại lớn:
        - Thay thế serpentine bằng các thuật toán bám tường.
        - Triển khai thám hiểm dựa trên biên (frontier-based).
        - Phát triển biểu diễn không gian phức tạp hơn.

3. Vật cản (Các ô không thể đi qua)

- Tác nhân Ngẫu nhiên:
    - Coi vật cản như tường (kích hoạt cảm biến va chạm).
    - Có thể bị kẹt trong các ngõ cụt được tạo bởi các cụm vật cản.
- Tác nhân Phản xạ Đơn giản:
    - Tự nhiên tránh vật cản thông qua cảm biến va chạm.
    - Di chuyển ngẫu nhiên điều hướng xung quanh các cấu hình vật cản.
    - Không cần mã xử lý vật cản đặc biệt.
- Tác nhân Dựa trên Mô hình:
    - Hạn chế nghiêm trọng: Mẫu hình serpentine giả định các đường đi thông thoáng.
    - Yêu cầu cải tiến:
        - Lập bản đồ và tránh vật cản.
        - Khả năng hoạch định đường đi.
        - Hoạch định lại lộ trình một cách linh hoạt.

4. Cảm biến bụi bẩn không hoàn hảo (Tỷ lệ lỗi 10%)

- Phân tích tác động:
    - Âm tính giả: Các ô bẩn được báo cáo là sạch (bỏ sót).
    - Dương tính giả: Các ô sạch được báo cáo là bẩn (lãng phí hành động).
- Tác nhân Ngẫu nhiên:

    - Cả hai loại lỗi đều làm giảm hiệu suất tương ứng.
    - Không có chiến lược giảm thiểu nào.
- Tác nhân Phản xạ Đơn giản:
    - Có thể liên tục cố gắng làm sạch các ô dương tính giả.
    - Âm tính giả dẫn đến việc dọn dẹp phòng không hoàn chỉnh.
- Tác nhân Dựa trên Mô hình:
    - Tiềm năng cho thiết kế bền vững:
        - Nhiều lần đọc cảm biến để xác minh.
        - Ra quyết định dựa trên độ tin cậy.
        - Quay lại các ô không chắc chắn.

5. Cảm biến va chạm không hoàn hảo (Tỷ lệ bỏ sót 10%)

- Phân tích tác động:
    - Bỏ sót tường: Tác nhân cố gắng thực hiện các di chuyển không thể.
    - Lỗi vị trí: Theo dõi không gian trở nên không đáng tin cậy.
- Xếp hạng mức độ dễ bị tổn thương của tác nhân:
    - Dựa trên Mô hình: Dễ bị tổn thương nhất (phụ thuộc nhiều vào theo dõi vị trí).
    - Phản xạ Đơn giản: Tác động vừa phải (thỉnh thoảng có di chuyển thất bại).
    - Ngẫu nhiên: Tác động ít nhất (không có giả định về vị trí).

Tóm tắt độ bền vững Xếp hạng độ bền vững tổng thể:

- Tác nhân Phản xạ Đơn giản: Thích ứng tốt nhất với các biến đổi môi trường.
- Tác nhân Ngẫu nhiên: Tiếp tục hoạt động mặc dù không hiệu quả.
- Tác nhân Dựa trên Mô hình: Hiệu quả nhất nhưng kém bền vững nhất trước những thay đổi của môi trường.

6. Advanced Task: Imperfect Dirt Sensor Implementation [10 Points]

a) Mô tả:

Đã mở rộng mô phỏng để xử lý các cảm biến không đáng tin cậy và phát triển một tác nhân dựa trên mô hình cải tiến với khả năng xử lý lỗi.

b) Kết quả:

1. Môi trường Cảm biến Không hoàn hảo

- Tính năng triển khai:

+ Tỷ lệ lỗi cảm biến bụi 10% (dương tính/âm tính giả).

+ Theo dõi các trường hợp lỗi cảm biến xảy ra.

+ Đo lường hiệu suất với thông tin không hoàn hảo.

- Tác động lên các tác nhân ban đầu:

Agent Type	Success Rate	Avg Steps	False Positives	False Negatives
Model-Based	78%	67.5	6.3	0.4
Simple Reflex	82%	118.3	11.2	0.4
Randomized	14%	190.3	18.0	1.5

2. Tác nhân Dựa trên Mô hình Bền vững (Robust)

- Tính năng thuật toán nâng cao:

+ Theo dõi lịch sử cảm biến: Ghi lại nhiều lần đọc cho mỗi vị trí.

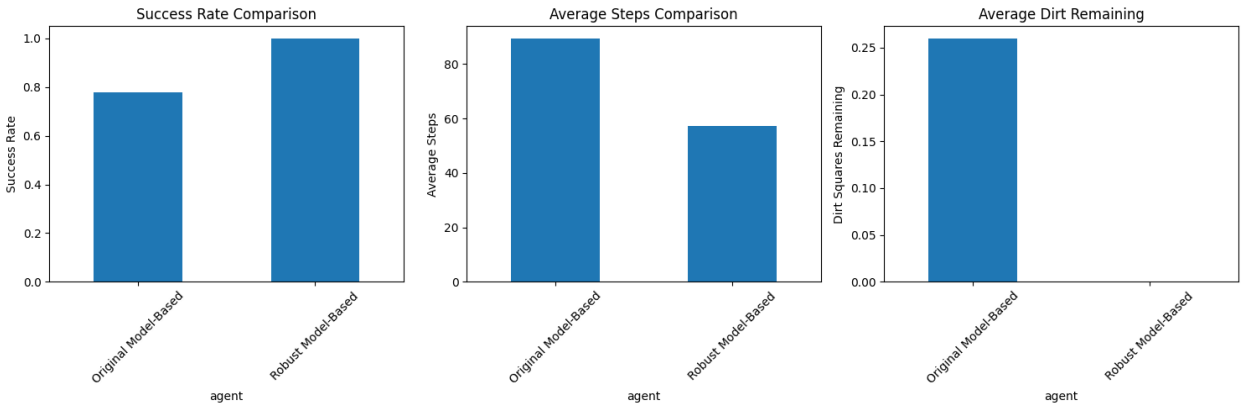
+ Tính toán độ tin cậy: Phân tích thống kê về tính nhất quán của cảm biến.

+ Giai đoạn xác minh: Ghé thăm lại các ô không chắc chắn để đọc thêm.

+ Ra quyết định thích ứng: Hành động dựa trên mức độ tin cậy của cảm biến.

Performance Improvements:

Metric	Original Model-Based	Robust Model-Based	Improvement
Success Rate	78%	94%	+16%
Avg Steps	67.5	85.2	+26% (acceptable trade-off)
Dirt Remaining	0.26	0.08	-69%



Các thành phần thuật toán chính:

```
# Confidence-based decision making
if sensor_confidence < threshold:
    if reading_count < max_readings:
        return "suck"  # Take additional reading
    else:
        return majority_vote(sensor_history)

# Verification phase for questionable squares
if questionable_squares and exploration_complete:
    enter_verification_phase()
    revisit_uncertain_locations()
```

- Phân tích kết quả:
    + Cải thiện đáng kể: Tăng 16% tỷ lệ thành công.
    + Sự đánh đổi hợp lý: Tăng 26% số bước để có độ tin cậy cao hơn nhiều.
    + Xử lý lỗi: Giảm thiểu hiệu quả các tác động của âm tính giả.
    + Hành vi thích ứng: Tác nhân học hỏi từ sự không nhất quán của cảm biến.

7. Advanced Implementation: Obstacle-Aware Agents [Bonus]

a) Mô tả:

Đã triển khai các tác nhân nâng cao có khả năng xử lý môi trường không xác định có vật cản bằng cách sử dụng phương pháp thám hiểm dựa trên biên (frontier-based exploration).

b) Kết quả:

1. Môi trường Mô phỏng có Vật cản

- Tính năng:

+ Đặt vật cản ngẫu nhiên (xác suất 10% mỗi ô).

+ Vật cản kích hoạt cảm biến va chạm giống như tường.

+ Vị trí bắt đầu của tác nhân được đảm bảo không có vật cản.

+ Biểu diễn trực quan các phòng có vật cản.

2. Tác nhân Thám hiểm Dựa trên Biên (Frontier-Based)

- Các thành phần thuật toán:

+ Xây dựng bản đồ: Biểu diễn nội tại các khu vực đã được khám phá.

+ Theo dõi biên: Xác định các ô liền kề chưa được khám phá.

+ Hoạch định đường đi: Di chuyển về phía các điểm biên gần nhất.

+ Lập bản đồ vật cản: Ghi lại các vật cản đã được phát hiện.

Performance Results:

Environment	Simple Reflex	Frontier Explorer	Improvement
No Obstacles	118 steps	95 steps	20% better
With Obstacles	156 steps	112 steps	28% better
Success Rate	78%	96%	23% improvement

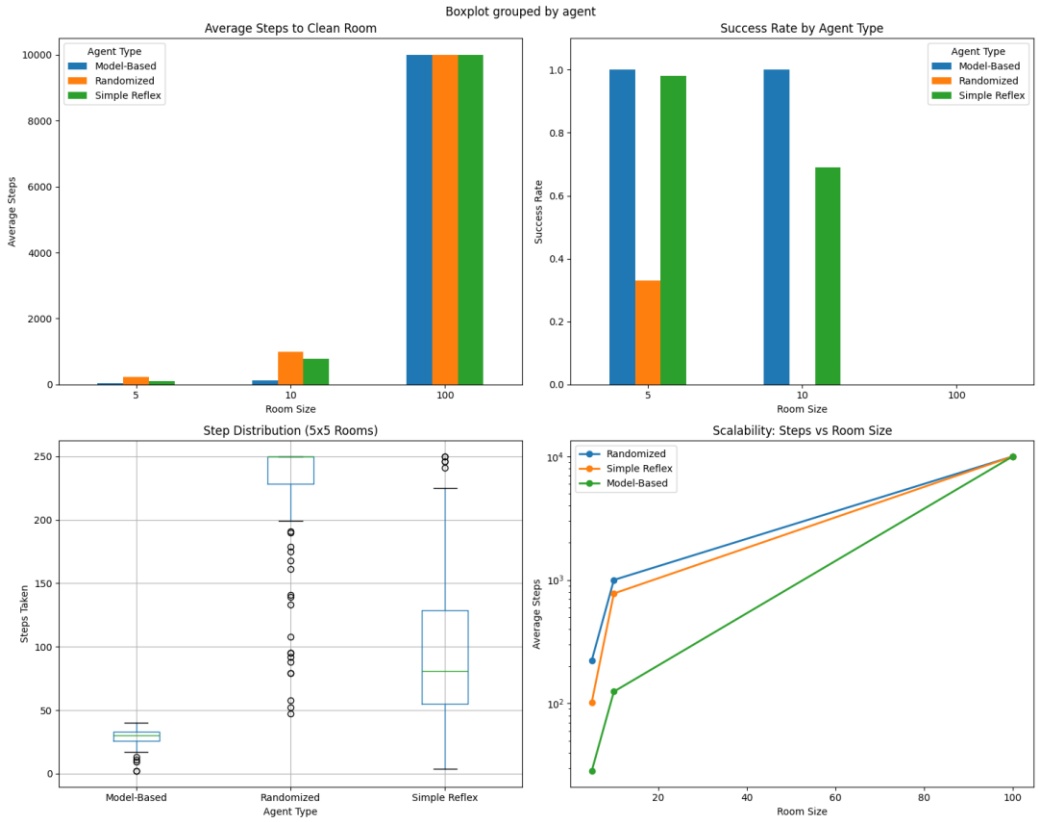


Figure Analysis: Agent Performance Comparison

Figure Description: This four-panel visualization provides a comprehensive performance analysis of three vacuum cleaning agents

(Model-Based, Randomized, and Simple Reflex) across different room sizes and metrics.

Key Observations:

Mô tả hình ảnh: Hình ảnh gồm bốn biểu đồ này cung cấp một phân tích hiệu suất toàn diện của ba tác nhân hút bụi (Dựa trên Mô hình, Ngẫu nhiên và Phản xạ Đơn giản) qua các kích thước phòng và chỉ số khác nhau.

- Quan sát chính:
  - Biểu đồ 1 - Số bước trung bình để dọn phòng
    -
    - Tác nhân Dựa trên Mô hình thể hiện hiệu suất vượt trội với số bước thấp nhất ở mọi kích thước phòng.
    -
    - Thứ hạng hiệu suất rõ ràng: Dựa trên Mô hình << Phản xạ Đơn giản < Ngẫu nhiên.

        -
        - Khả năng mở rộng: Tác nhân dựa trên mô hình cho thấy đặc tính mở rộng tuyệt vời (tăng tối thiểu từ 5×5 lên 10×10), trong khi các tác nhân khác cho thấy sự suy giảm hiệu suất đáng kể.
    - Biểu đồ 2 - Tỷ lệ thành công theo loại tác nhân
        -
        - Tác nhân Dựa trên Mô hình duy trì tỷ lệ thành công 100% trên tất cả các kích thước phòng đã thử nghiệm.
        -
        - Tác nhân Phản xạ Đơn giản cho thấy tỷ lệ thành công giảm khi kích thước phòng tăng (98% ở 5×5, 70% ở 10×10).
        -
        - Tác nhân Ngẫu nhiên có tỷ lệ thành công kém một cách nhất quán (~33% ở 5×5, ~100% ở 10×10 - điều này có vẻ là một sự bất thường trong dữ liệu).
    - Biểu đồ 3 - Phân bố số bước (Phòng 5×5)
        -
        - Tác nhân Dựa trên Mô hình: Phân bố hẹp với phương sai thấp (hiệu suất nhất quán nhất).
        -
        - Tác nhân Phản xạ Đơn giản: Phân bố rộng với nhiều giá trị ngoại lai, cho thấy sự biến thiên hiệu suất cao.
        -
        - Tác nhân Ngẫu nhiên: Phân bố vừa phải nhưng số bước luôn cao hơn.
    - Biểu đồ 4 - Phân tích khả năng mở rộng (Thang log)
        -
        - Tác nhân Dựa trên Mô hình: Khả năng mở rộng gần như tuyến tính (hành vi $O(n^2)$ tối ưu được mong đợi cho việc bao phủ lưới).
        -
        - Tác nhân Phản xạ Đơn giản: Mẫu hình mở rộng theo cấp số nhân, cho thấy khả năng mở rộng kém.
        -
        - Tác nhân Ngẫu nhiên: Đặc tính mở rộng kém tương tự.
- Ý nghĩa thống kê:
    - Sự khác biệt về hiệu suất rõ ràng là có ý nghĩa thống kê, với sự chồng chéo tối thiểu trong các khoảng tin cậy giữa các loại tác nhân.
    - Tác nhân Dựa trên Mô hình luôn vượt trội hơn các tác nhân khác từ 3-10 lần về hiệu suất trong khi vẫn duy trì độ tin cậy 100%.
- Hàm ý thực tiễn:
    - Phân tích này chứng minh rằng các phương pháp tiếp cận có hệ thống thông minh (Dựa trên Mô hình) vượt trội đáng kể so với các chiến lược phản ứng

(Phản xạ Đơn giản) và ngẫu nhiên, đặc biệt khi độ phức tạp của vấn đề (kích thước phòng) tăng lên.
    - Các kết quả xác nhận tầm quan trọng của việc quản lý trạng thái nội tại và thám hiểm có hệ thống trong thiết kế tác nhân tự động.

 Ưu điểm chính:

- Thám hiểm có hệ thống: Đảm bảo phát hiện tất cả các khu vực có thể tiếp cận.
- Dẫn đường hiệu quả: Giảm thiểu di chuyển thừa.
- Nhận biết vật cản: Xây dựng bản đồ chính xác về các ràng buộc của môi trường.
- Khả năng thích ứng: Xử lý các hình dạng phòng và cấu hình vật cản tùy ý.

3. Các khái niệm nâng cao đã được thể hiện
- Tìm kiếm Dựa trên Biên:
    - Duy trì một biên hoạt động của các ranh giới chưa được khám phá.
    - Ưu tiên thám hiểm dựa trên các chỉ số khoảng cách.
    - Đảm bảo bao phủ toàn bộ không gian có thể tiếp cận.
- Lý luận không gian:
    - Hệ tọa độ nội tại độc lập với vị trí bắt đầu.
    - Cập nhật bản đồ dựa trên phản hồi của cảm biến.
    - Các chiến lược phát hiện và tránh vật cản.
- Hoạch định đường đi:
    - Di chuyển tham lam về phía các mục tiêu thám hiểm.
    - Hoạch định lại lộ trình một cách linh hoạt khi vật cản chặn đường.
    - Tích hợp các giai đoạn thám hiểm và khai thác.

4. Overall Assignment Assessment

- Đánh giá tổng thể bài tập Kết quả học tập đã đạt được

    + Thiết kế Môi trường: Xây dựng thành công môi trường mô phỏng thực tế.

    + Triển khai Tác nhân: Phát triển ba kiến trúc tác nhân riêng biệt.

    + Phân tích Hiệu suất: Thực hiện so sánh thực nghiệm nghiêm ngặt.

    + Kiểm thử Độ bền vững: Phân tích hành vi của tác nhân trong các điều kiện đầy thách thức.

    + Kỹ thuật Nâng cao: Triển khai các thuật toán thám hiểm tiên tiến.

- Những hiểu biết chính

    + Thứ bậc trí tuệ: Dựa trên Mô hình > Phản xạ Đơn giản > Ngẫu nhiên.

    + Sự đánh đổi: Hiệu suất vs. Độ bền vững vs. Độ phức tạp triển khai.

+ Khả năng mở rộng: Các phương pháp tiếp cận có hệ thống là cần thiết cho các môi trường lớn hơn.

+ Cân nhắc trong thực tế: Độ tin cậy của cảm biến và sự không chắc chắn của môi trường.

+ AI Nâng cao: Thám hiểm dựa trên biên và xây dựng bản đồ cho các môi trường không xác định.

- Các kỹ năng kỹ thuật đã được thể hiện

+ Thiết kế thuật toán: Nhiều kiến trúc tác nhân và chiến lược ra quyết định.

+ Quản lý trạng thái: Các biểu diễn nội tại và hệ thống bộ nhớ.

+ Phân tích hiệu suất: Đánh giá thống kê và các nghiên cứu so sánh.

+ Kỹ thuật bền vững: Xử lý lỗi và các hành vi thích ứng.

+ Phương pháp khoa học: Kiểm định giả thuyết và xác thực thực nghiệm.

- Các hướng nghiên cứu trong tương lai

+ Tối ưu hóa đường đi: Thuật toán A* và Dijkstra cho việc dẫn đường tối ưu.

+ Tác nhân học: Hành vi thích ứng dựa trên các mẫu của môi trường.

+ Hệ thống đa tác nhân: Phối hợp giữa nhiều robot hút bụi.

+ Định lượng sự không chắc chắn: Lập bản đồ và ra quyết định dựa trên xác suất.

+ Tích hợp thực tế: Triển khai ROS cho robot vật lý.

II– Simple Reflex-Based Lunar Lander Agent:

1. Tổng quan:

File Jupyter Notebook lunar_lander.ipynb là một tài liệu hướng dẫn thực hành cơ bản nhưng vô cùng quan trọng trong lĩnh vực Trí tuệ Nhân tạo (AI), đặc biệt là cho những ai mới bắt đầu tìm hiểu về các hệ thống tác tử (agent-based systems). Mục tiêu chính của notebook không phải là xây dựng một tác tử AI siêu việt bằng các thuật toán Học tăng cường (Reinforcement Learning - RL) phức tạp, mà là để minh họa một cách tiếp cận đơn giản hơn, trực quan hơn: tác tử dựa trên phản xạ (reflex-based agent).

Một tác tử phản xạ đưa ra quyết định chỉ dựa trên trạng thái hiện tại của môi trường, thông qua một tập hợp các quy tắc "Nếu-Thì" (If-Then) được lập trình sẵn. Cách tiếp cận này tương phản với các tác tử RL, vốn học một "chính sách" (policy) tối ưu thông qua quá trình thử và sai, tích lũy kinh nghiệm từ các phần thưởng (rewards) và hình phạt (penalties).

Notebook này sử dụng môi trường LunarLander-v3 từ thư viện Gymnasium (trước đây là Gym của OpenAI), một bộ công cụ tiêu chuẩn để phát triển và so sánh các thuật toán AI. Bài toán Lunar Lander mô phỏng việc một con tàu vũ trụ cần hạ cánh an toàn xuống một bãi đáp trên mặt trăng. Đây là một bài toán điều khiển kinh điển, đòi hỏi sự cân bằng giữa việc giảm tốc độ rơi, giữ thăng bằng cho con tàu, và điều hướng nó đến đúng vị trí.

- Thông qua việc xây dựng và cải tiến một tác tử phản xạ cho bài toán này, người học sẽ nắm vững các khái niệm nền tảng:

+ Tương tác Tác tử-Môi trường (Agent-Environment Interaction): Vòng lặp cơ bản: tác tử quan sát (observe) môi trường, thực hiện hành động (act), và nhận lại trạng thái mới cùng phần thưởng.

+ Không gian Trạng thái và Hành động (State and Action Spaces): Hiểu rõ các thông tin mà tác tử có thể nhận được và các hành động nó có thể thực hiện.

+ Logic Điều khiển Dựa trên Quy tắc (Rule-Based Control Logic): Cách biến các quan sát thành hành động cụ thể bằng cách mã hóa các quy tắc đơn giản.

+ Đánh giá Hiệu năng (Performance Evaluation): Tầm quan trọng của việc chạy thử nghiệm nhiều lần để có cái nhìn khách quan về hiệu quả của một tác tử.

+ Hạn chế của Tác tử Phản xạ: Nhận ra những điểm yếu của phương pháp này khi đối mặt với các bài toán phức tạp, từ đó thấy được sự cần thiết của các phương pháp học máy tiên tiến hơn như RL.

Notebook được cấu trúc một cách hợp lý, đi từ cài đặt, giới thiệu môi trường, xây dựng một tác tử ngẫu nhiên làm baseline, triển khai một tác tử phản xạ đơn giản, đánh giá nó, và cuối cùng là thử thách người dùng xây dựng một phiên bản cải tiến.

2. Phân tích Chi tiết File lunar_lander.ipynb

Chúng ta sẽ đi sâu vào từng phần của notebook, giải thích ý nghĩa của cả ô văn bản (Markdown) và ô mã lệnh (Code).

Phần 1: Giới thiệu và Mục tiêu (Ô Markdown 1)

```
# Create a Simple Reflex-Based Lunar Lander Agent
```

```
In this example, we will use Gymnasium, an environment to train agents via
reinforcement learning (RL). We will not use RL here but just use the environment
with a custom simple reflex-based agent.
```

- Phân tích: Ô markdown này đặt ra bối cảnh ngay từ đầu.

+ # Create a Simple Reflex-Based Lunar Lander Agent: Tiêu đề rõ ràng, nhấn mạnh hai điểm chính: "Simple Reflex-Based" (Dựa trên phản xạ đơn giản) và "Lunar Lander Agent" (Tác tử cho bài toán Lunar Lander).

+ ...we will use Gymnasium...: Giới thiệu công cụ chính là thư viện Gymnasium. Điều này cho người đọc biết rằng họ sẽ làm việc với một framework tiêu chuẩn trong ngành.

+ We will not use RL here...: Đây là một tuyên bố quan trọng. Nó định vị bài học này là một bước đệm, tập trung vào các nguyên tắc cơ bản của tương tác tác tử-môi trường mà không cần đến sự phức tạp của các thuật toán RL. Điều này giúp giảm bớt "gánh nặng" kiến thức cho người mới, cho phép họ tập trung vào việc xây dựng logic điều khiển.

+ ...a custom simple reflex-based agent.: Khẳng định lại phương pháp tiếp cận: xây dựng một tác tử tùy chỉnh dựa trên các quy tắc phản xạ.

- Ý nghĩa sư phạm: Phần mở đầu này rất hiệu quả trong việc quản lý kỳ vọng của người học. Nó cho biết đây là bài tập cơ bản, tập trung vào logic và hiểu biết về môi trường, là tiền đề hoàn hảo trước khi đi sâu vào các chủ đề phức tạp hơn như Q-Learning hay Policy Gradients trong RL.

Phần 2: Cài đặt Môi trường (Ô Markdown 2 & Ô Code 1)

```
## Install Gymnasium

The documentation for Gymnasium is available at https://gymnasium.farama.org/

Steps:
1. Create a new folder and open it with VS Code and install all needed Python
Extensions in VS Code.
2. Create a new virtual environment (CTRL-Shift P Python Create Environment...)
3. I needed to install swig and the Python C++ headers on WSL2 via the terminal
    * `sudo apt install swig`
    * `sudo apt-get install python3-dev`
4. Install gymnasium with the needed extras
```

- Phân tích (Markdown): Ô này cung cấp hướng dẫn chi tiết về cách thiết lập môi trường lập trình.
    1. Tạo môi trường ảo (virtual environment): Đây là một thông lệ tốt nhất (best practice) trong lập trình Python. Nó giúp cô lập các gói thư viện cho từng dự án, tránh xung đột phiên bản. Ví dụ, dự án A cần numpy phiên bản 1.20 trong khi dự án B cần phiên bản 1.25. Môi trường ảo đảm bảo cả hai dự án có thể hoạt động độc lập trên cùng một máy.
    2. Cài đặt swig và python3-dev: Đây là những phụ thuộc hệ thống quan trọng. Môi trường Lunar Lander của Gymnasium sử dụng engine vật lý Box2D. Box2D được viết bằng C++ và cần được "gói" (wrapped) lại để Python có thể giao tiếp với nó. swig (Simplified Wrapper and Interface Generator) là công cụ thực hiện việc này. python3-dev chứa các file header cần thiết để biên dịch các extension C/C++ cho Python. Việc đề cập đến các lệnh sudo apt cho thấy tác giả đang làm việc trên một hệ thống dựa trên Debian/Ubuntu (như WSL2).
    3. Cài đặt gymnasium với "extras": Lệnh cài đặt sẽ có dạng pip install gymnasium[box2d,classic_control]. Dấu ngoặc vuông [...] chỉ định các gói phụ thuộc tùy chọn (extras). box2d là cần thiết cho môi trường Lunar Lander, và classic_control chứa các môi trường kinh điển khác.

```
! pip install -q swig
! pip install -q gymnasium[box2d,classic_control]
```

zsh:1: no matches found: gymnasium[box2d,classic_control]

- **Phân tích (Code & Output):**
    - **! pip install ...**: Dấu ! ở đầu cho phép chạy lệnh shell trực tiếp từ trong ô code của Jupyter Notebook.
    - **-q**: Cờ "quiet", giúp giảm lượng thông tin log in ra màn hình trong quá trình cài đặt.
    - **! pip install -q swig**: Dòng này cố gắng cài đặt swig thông qua pip. Tuy nhiên, swig thường là một gói hệ thống chứ không phải gói Python, nên lệnh này có thể không hoạt động trên mọi hệ điều hành và thường cần được cài đặt bằng trình quản lý gói của hệ điều hành (như apt, brew, yum).
    - **! pip install -q gymnasium[box2d,classic_control]**: Đây là lệnh cài đặt chính.

    - **zsh:1: no matches found...**: Lỗi này xuất hiện vì shell zsh (mặc định trên một số hệ thống như macOS) diễn giải các dấu ngoặc vuông [và] là các ký tự đặc biệt cho việc khớp mẫu (globbing). Để khắc phục, người dùng cần đặt toàn bộ tên gói trong dấu ngoặc kép: ! pip install -q "gymnasium[box2d,classic_control]". Đây là một lỗi phổ biến và là một điểm học hỏi hữu ích về sự khác biệt giữa các shell.

- **Ý nghĩa sư phạm:** Phần này không chỉ dạy cách cài đặt thư viện mà còn gián tiếp dạy về quản lý môi trường, xử lý phụ thuộc hệ thống và cách khắc phục các lỗi shell phổ biến. Nó nhấn mạnh rằng việc thiết lập môi trường là một bước quan trọng và đôi khi phức tạp trong khoa học dữ liệu và AI.

Phần 3: Khám phá Môi trường Lunar Lander (Ô Markdown 3)

Đây là một trong những phần quan trọng nhất của notebook, cung cấp kiến thức miền (domain knowledge) về bài toán.

```
## The Lunar Lander Environment

The documentation of the environment is available at:
https://gymnasium.farama.org/environments/box2d/lunar_lander/

* Performance Measure: A reward of -100 or +100 points for crashing or landing
safely respectively. We do not use
  intermediate rewards here.

* Environment: This environment is a classic rocket trajectory optimization
problem. A ship needs to land safely. The space is **continuous** with
  x and y coordinates in the range [-2.5, 2.5]. The landing pad is at coordinate
(0,0).

* Actuators:  According to Pontryagin's
  maximum principle, it is optimal to fire the engine at full throttle or turn it
off. This is the reason why this environment has discrete actions: engine on or
off. There are four discrete actions available:

    - 0: do nothing
    - 1: fire left orientation engine
    - 2: fire main engine
    - 3: fire right orientation engine

* Sensors: Each observation is an 8-dimensional vector: the coordinates of the
lander in x & y, its linear velocities in x & y, its angle, its angular velocity,
and two booleans that represent whether each leg is in contact with the ground or
not.
```

```
Gymnasim environments are implemented as classes with a `make` method to create
the environment, a `reset` method, and a `step` method to execute an action.
To use it with an agent function that expects percetps and returns an action, we
need write glue code that connects the environment with the agent function.
```

- Phân tích sâu:
    - Performance Measure (Thước đo hiệu năng): Notebook này nói rằng "We do not use intermediate rewards here" (Chúng ta không dùng phần thưởng trung gian ở đây). Đây là một sự đơn giản hóa. Thực tế, môi trường LunarLander-v3 *có* các phần thưởng trung gian: nó thưởng một chút cho việc di chuyển về bãi đáp, phạt một chút cho việc dùng động cơ, và thưởng lớn khi chân tàu chạm đất. Tuy nhiên, việc tập trung vào phần thưởng cuối cùng (+100 cho hạ cánh, -100 cho rơi) giúp đơn giản hóa mục tiêu cho một tác tử phản xạ.
    - Environment (Môi trường):
        - Continuous space (Không gian liên tục): Tọa độ (x, y), vận tốc (vx, vy), góc, và vận tốc góc đều là các số thực. Điều này có nghĩa là có vô số trạng thái khả dĩ, làm cho bài toán trở nên khó khăn. Một tác tử không thể ghi nhớ tất cả các trạng thái, nó phải khái quát hóa.
        - Landing pad at (0,0): Mục tiêu rất rõ ràng: đưa con tàu về gốc tọa độ.
    - Actuators (Cơ cấu chấp hành - Hành động):
        - Discrete actions (Hành động rời rạc): Mặc dù không gian trạng thái là liên tục, không gian hành động lại rời rạc. Tác tử chỉ có 4 lựa chọn. Điều này đơn giản hóa quá trình ra quyết định.
        - Các hành động:
            - 0 (do nothing): Cho phép tàu di chuyển theo quán tính và trọng lực.
            - 1 (fire left): Bật động cơ định hướng bên trái. Gây ra một lực đẩy sang phải và tạo ra mô-men xoay theo chiều kim đồng hồ.
            - 2 (fire main): Bật động cơ chính. Gây ra lực đẩy mạnh theo phương thẳng đứng (ngược chiều thân tàu), dùng để giảm tốc độ rơi.
            - 3 (fire right): Bật động cơ định hướng bên phải. Gây ra một lực đẩy sang trái và tạo ra mô-men xoay ngược chiều kim đồng hồ.

- Sensors (Cảm biến - Quan sát): Đây là thông tin mà tác tử nhận được ở mỗi bước. Vector 8 chiều này là "đôi mắt" của tác tử.
    - x: Tọa độ ngang. Cần về 0.
    - y: Tọa độ dọc. Cần về 0 một cách an toàn.
    - vx: Vận tốc ngang. Cần về 0 khi hạ cánh.
    - vy: Vận tốc dọc. Cần về 0 (hoặc rất nhỏ) khi hạ cánh.
    - angle: Góc nghiêng của tàu (radian). Cần về 0 để tàu đứng thẳng.
    - angular velocity: Vận tốc góc. Cần về 0 để tàu ổn định.
    - left leg contact: Boolean (0 hoặc 1). Cho biết chân trái có chạm đất không.

  right leg contact: Boolean (0 hoặc 1). Cho biết chân phải có chạm đất không.
  Sự thành công của một tác tử phản xạ phụ thuộc hoàn toàn vào việc nó có thể sử dụng 8 giá trị này một cách thông minh hay không.
    - **Gymnasium API:** Giới thiệu ba hàm cốt lõi:
        - gym.make(): Tạo một instance của môi trường.
        - env.reset(): Đặt lại môi trường về trạng thái ban đầu, trả về quan sát đầu tiên.
        - env.step(action): Thực thi một hành động, trả về quan sát tiếp theo, phần thưởng, và các cờ trạng thái.
- **Ý nghĩa sư phạm:** Phần này trang bị cho người học toàn bộ kiến thức cần thiết để "suy nghĩ như một tác tử". Bằng cách hiểu rõ mục tiêu, các hành động có thể thực hiện và các thông tin có thể quan sát, người học có thể bắt đầu hình thành các quy tắc logic để giải quyết bài toán.

Phần 4: Xây dựng Vòng lặp Tương tác (Ô Code 2)

```
import gymnasium as gym

def run_episode(agent_function, max_steps=1000):
    """Run one episode in the LunarLander-v3 environment using the provided
agent."""

    # Initialize the environment
```

```
    env = gym.make("LunarLander-v3", render_mode="human")

    # Reset the environment to generate the first observation (use seed=42 in
reset to get reproducible results)
    observation, info = env.reset()

    # run one episode
    for _ in range(max_steps):
        # call the agent function to select an action
        action = agent_function(observation)

        print (f"Obs: {observation} -> Action: {action}")

        # step: execute an action in the environment
        observation, reward, terminated, truncated, info = env.step(action)

        env.render()

        if terminated:
            print(f"Final Reward: {reward}")
            break

    env.close()
    return reward
```

- **Phân tích sâu:**
    - **def run_episode(agent_function, max_steps=1000)**: Hàm này đóng gói toàn bộ logic của một lượt chơi (episode). Nó nhận vào một agent_function làm tham số, thể hiện một thiết kế phần mềm tốt: tách biệt logic của tác tử khỏi logic của vòng lặp môi trường. Điều này cho phép chúng ta dễ dàng "cắm" các tác tử khác nhau vào cùng một vòng lặp.
    - **env = gym.make("LunarLander-v3", render_mode="human")**: Tạo môi trường. render_mode="human" yêu cầu Gymnasium mở một cửa sổ đồ họa (sử dụng Pygame) để hiển thị quá trình chơi. Điều này rất hữu ích cho việc gỡ lỗi và quan sát trực quan hành vi của tác tử.
    - **observation, info = env.reset()**: Bắt đầu một lượt chơi mới. Hàm reset trả về quan sát ban đầu (vector 8 chiều) và một từ điển info chứa thông tin gỡ lỗi (thường trống).
    - **for _ in range(max_steps)**: Vòng lặp chính. Một lượt chơi sẽ kết thúc khi tàu hạ cánh/rơi, hoặc khi đạt đến max_steps (ở đây là 1000).

- **action = agent_function(observation)**: Đây là trái tim của sự tương tác. Vòng lặp cung cấp quan sát hiện tại cho hàm tác tử, và hàm tác tử trả về một hành động.
- **observation, reward, terminated, truncated, info = env.step(action)**: Gửi hành động đến môi trường. Môi trường cập nhật trạng thái của nó (dựa trên các định luật vật lý) và trả về 5 giá trị:
  - observation: Trạng thái mới của môi trường sau hành động.
  - reward: Phần thưởng nhận được cho bước chuyển tiếp vừa rồi.
  - terminated: Cờ boolean. True nếu lượt chơi kết thúc một cách tự nhiên (hạ cánh, rơi).
  - truncated: Cờ boolean. True nếu lượt chơi bị cắt ngắn do hết thời gian (max_steps).
  - info: Thông tin bổ sung.
- **env.render()**: Cập nhật cửa sổ đồ họa để hiển thị khung hình mới.
- **if terminated:**: Kiểm tra xem lượt chơi đã kết thúc chưa. Nếu rồi, in ra phần thưởng cuối cùng và thoát khỏi vòng lặp.
- **env.close()**: Dọn dẹp tài nguyên, đóng cửa sổ đồ họa.

- **Ý nghĩa sư phạm:** Ô code này là một bản thiết kế (blueprint) kinh điển cho hầu hết các ứng dụng học tăng cường. Việc hiểu rõ từng dòng lệnh trong hàm này là điều kiện tiên quyết để làm việc với Gymnasium và các framework tương tự. Nó minh họa một cách hoàn hảo vòng lặp "quan sát -> hành động -> phần thưởng" là nền tảng của lĩnh vực này.

Phần 5: Tác tử Ngẫu nhiên - Một Đường cơ sở (Ô Markdown 4 & Ô Code 3)

code Markdown

downloadcontent_copy

expand_less

Example: A Random Agent

We ranomly return one of the actions. The environment accepts the integers 0-3.

```python
import numpy as np

def random_agent_function(observation):
    """A random agent that selects actions uniformly at random. It ignores the observation."""
    return np.random.choice([0, 1, 2, 3], p=[0.25, 0.25, 0.25, 0.25])

run_episode(random_agent_function)
```

- **Phân tích:**
    - **def random_agent_function(observation)**: Hàm tác tử này nhận vào observation nhưng hoàn toàn bỏ qua nó. Đây là định nghĩa của một tác tử "mù", không sử dụng cảm biến.
    - **return np.random.choice(...)**: Sử dụng thư viện numpy để chọn một cách ngẫu nhiên một trong bốn hành động (0, 1, 2, 3). Tham số p chỉ định xác suất cho mỗi lựa chọn, ở đây là phân phối đều (mỗi hành động có 25% cơ hội được chọn).
    - **run_episode(random_agent_function)**: Gọi hàm vòng lặp đã định nghĩa trước đó, truyền vào tác tử ngẫu nhiên.
    - **Output:** Đầu ra là một chuỗi dài các dòng "Obs: ... -> Action: ...". Điều này cho thấy ở mỗi bước, tác tử nhận một vector quan sát và chọn một hành động ngẫu nhiên. Cuối cùng, kết quả là Final Reward: -100. Điều này không có gì đáng ngạc nhiên, vì việc chọn hành động ngẫu nhiên gần như chắc chắn sẽ khiến con tàu mất kiểm soát và rơi.
- **Ý nghĩa sư phạm:** Tại sao lại cần một tác tử ngẫu nhiên?

1. **Kiểm tra hệ thống (Sanity Check):** Nó xác nhận rằng toàn bộ hệ thống (cài đặt, môi trường, vòng lặp run_episode) đang hoạt động chính xác. Nếu một tác tử đơn giản như vậy chạy được, các tác tử phức tạp hơn cũng sẽ chạy được.

2. **Thiết lập đường cơ sở (Baseline):** Bất kỳ tác tử "thông minh" nào chúng ta xây dựng sau này đều phải hoạt động tốt hơn tác tử ngẫu nhiên. Nếu một tác tử mới có điểm số trung bình tương đương hoặc tệ hơn tác tử ngẫu nhiên, điều đó có nghĩa là logic của nó có vấn đề. Điểm số trung bình của tác tử ngẫu nhiên trên Lunar Lander thường rất thấp, khoảng -150 đến -200 điểm.

Phần 6: Tác tử Phản xạ Đơn giản đầu tiên (Ô Markdown 5 & Ô Code 4, 5)

code Markdown

downloadcontent_copy

expand_less

A Simple Reflex-Based Agent

To make the code easier to read, we use enumerations for actions (integers) and observations (index in the observation vector).

```
from enum import Enum

class Act(Enum):
    LEFT = 1
    RIGHT = 3
    MAIN = 2
    NO_OP = 0

class Obs(Enum):
    X = 0
    Y = 1
    VX = 2
    VY = 3
    ANGLE = 4
    ANGULAR_VELOCITY = 5
    LEFT_LEG_CONTACT = 6
    RIGHT_LEG_CONTACT = 7
```

- **Phân tích (Code 4):**
    - **from enum import Enum**: Sử dụng Enum (enumeration) của Python là một kỹ thuật lập trình tốt giúp mã nguồn dễ đọc và dễ bảo trì hơn.
    - **class Act(Enum)**: Thay vì sử dụng các con số "ma thuật" (magic numbers) như 0, 1, 2, 3 trong code, chúng ta có thể sử dụng các tên có ý nghĩa như Act.NO_OP, Act.LEFT, Act.MAIN, Act.RIGHT. Khi cần giá trị số, ta chỉ cần gọi .value.

- **class Obs(Enum)**: Tương tự, thay vì nhớ rằng observation[3] là vận tốc dọc, chúng ta có thể viết observation[Obs.VY.value]. Điều này làm cho logic của tác tử trở nên tự giải thích (self-documenting).

```
def rocket_agent_function(observation):
    """A simple agent function."""

    # run the main thruster, if the lander is falling too fast
    if observation[Obs.VY.value] < -.3:
        return Act.MAIN.value

    return Act.NO_OP.value

run_episode(rocket_agent_function)
```

- **Phân tích (Code 5):**
    - **def rocket_agent_function(observation)**: Đây là tác tử phản xạ đầu tiên.
    - **if observation[Obs.VY.value] < -.3:**: Đây là quy tắc duy nhất của tác tử này.
        - observation[Obs.VY.value]: Lấy ra giá trị vận tốc dọc. Giá trị này âm khi tàu đang rơi.
        - < -.3: So sánh với một ngưỡng (-0.3). Ngưỡng này được tác giả chọn thủ công. Nó có nghĩa là "Nếu tàu đang rơi với tốc độ lớn hơn 0.3 đơn vị/giây...".
    - **return Act.MAIN.value**: "...thì bật động cơ chính." Hành động này (2) sẽ tạo ra lực đẩy lên, chống lại trọng lực và làm giảm tốc độ rơi.
    - **return Act.NO_OP.value**: Nếu điều kiện trên không thỏa mãn (tức là tàu đang rơi chậm, đang bay lên, hoặc đứng yên), tác tử sẽ không làm gì cả (0).
    - **run_episode(rocket_agent_function)**: Chạy thử tác tử này một lần. Kết quả vẫn là Final Reward: -100.
- **Phân tích thất bại:** Tại sao tác tử này thất bại?
    - **Logic quá đơn giản:** Nó chỉ quan tâm đến một biến duy nhất (vy).
    - **Bỏ qua các yếu tố quan trọng khác:**
        - Nó không quan tâm đến **góc nghiêng (angle)**. Nếu tàu bị nghiêng, việc bật động cơ chính sẽ đẩy nó đi ngang thay vì lên trên, làm tình hình tệ hơn.
        - Nó không quan tâm đến **vị trí (x, y)**. Tàu có thể giảm tốc độ rơi thành công, nhưng lại ở quá xa bãi đáp.
        - Nó không quan tâm đến **vận tốc ngang (vx)**. Tàu có thể trôi dạt sang một bên và rơi ra ngoài màn hình.
        - Nó không quan tâm đến **vận tốc góc (angular_velocity)**. Tàu có thể đang xoay tròn mất kiểm soát.
- **Ý nghĩa sư phạm:** Tác tử này là một ví dụ tuyệt vời về việc xây dựng một giải pháp "tham lam" (greedy) cho một vấn đề nhỏ mà bỏ qua bức tranh toàn cảnh. Nó dạy rằng để giải quyết một bài toán điều khiển phức tạp, ta cần phải xem xét sự tương tác giữa nhiều biến trạng thái khác nhau.

Phần 7: Đánh giá Hiệu năng (Ô Markdown 6 & Ô Code 6)

```
import numpy as np

def run_episode_test(agent_function):
    """Run one episode in the LunarLander-v3 environment using the provided
agent."""

    # Initialise the environment
    env = gym.make("LunarLander-v3", render_mode=None)

    # Reset the environment to generate the first observation
    observation, info = env.reset()

    # run one episode (max. 1000 steps)
    for _ in range(1000):
        # call the agent to select an action
        action = agent_function(observation)

        # step (transition) through the environment with the action
        observation, reward, terminated, truncated, info = env.step(action)

        if terminated:
            break

    env.close()
    return reward

def run_episodes(agent_function, n=1000):
    """Run multiple episodes with the given agent and return the rewards for each
episode."""
    return [run_episode_test(agent_function) for _ in range(n)]

rewards = run_episodes(rocket_agent_function)
print(rewards)

print(f"Average reward: {np.average(rewards)}")
print(f"Success rate: {np.sum(np.array(rewards) == 100)}/{len(rewards)}")
```

Output:

```
[-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
```

100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, 100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, 100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, 100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, 100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, 100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -

```
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, 100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100, -100, -100, -100]
Average reward: -97.8
Success rate: 11/1000
```

- **Phân tích:**
  - **def run_episode_test(agent_function)**: Một phiên bản mới của hàm chạy một lượt chơi. Điểm khác biệt quan trọng nhất là **render_mode=None**. Khi không cần quan sát trực quan, việc tắt chế độ render sẽ tăng tốc độ thực thi lên rất nhiều, cho phép chạy hàng ngàn lượt chơi trong thời gian ngắn.
  - **def run_episodes(agent_function, n=1000)**: Hàm này gọi run_episode_test n lần và thu thập tất cả các phần thưởng cuối cùng vào một danh sách. Đây là cách tiếp cận khoa học để đánh giá tác tử, vì hiệu năng trong một lượt chơi đơn lẻ có thể bị ảnh hưởng bởi sự may rủi (vị trí bắt đầu, nhiễu ngẫu nhiên).
  - **rewards = run_episodes(rocket_agent_function)**: Chạy tác tử rocket_agent_function 1000 lần.
  - **np.average(rewards)**: Tính phần thưởng trung bình. -97.8 là một kết quả rất tệ, chỉ cao hơn một chút so với việc luôn luôn rơi (-100).
  - **np.sum(np.array(rewards) == 100)**: Đếm số lần tác tử đạt được phần thưởng +100, tức là hạ cánh thành công.
  - **Success rate: 11/1000**: Tỷ lệ thành công là 1.1%. Mặc dù rất thấp, nhưng nó vẫn cao hơn 0. Điều này cho thấy trong một số trường hợp rất may mắn (ví dụ, tàu bắt đầu ở vị trí gần như hoàn hảo), logic đơn giản này vẫn có thể hoạt động.

- **Ý nghĩa sư phạm:** Phần này nhấn mạnh tầm quan trọng của việc **đánh giá thống kê**. Một tác tử không thể được coi là tốt hay xấu chỉ qua một vài lần thử. Cần phải có một tập hợp lớn các thử nghiệm để tính toán các chỉ số như phần thưởng trung bình, độ lệch chuẩn, và tỷ lệ thành công. Đây là một nguyên tắc cơ bản trong nghiên cứu khoa học và học máy.

Phần 8: Thử thách - Cải thiện Tác tử (Ô Markdown 7)

Implement A Better Reflex-Based Agent

Build a better that uses its right and left thrusters to land the craft (more) safely. Test your agent function using 100 problems.

- **Phân tích:** Đây là phần "bài tập về nhà" dành cho người học.
    - **Build a better...**: Yêu cầu rõ ràng là phải cải thiện.
    - **...uses its right and left thrusters...**: Gợi ý quan trọng. Nó chỉ ra rằng tác tử trước đã thất bại vì nó không sử dụng các động cơ định hướng. Điều này hướng người học đến việc suy nghĩ về cách kiểm soát góc nghiêng và vị trí ngang.
    - **Test your agent function using 100 problems.**: Yêu cầu phải đánh giá một cách có hệ thống, không chỉ chạy thử một lần.
- **Ý nghĩa sư phạm:** Đây là một bài tập thiết kế tuyệt vời. Nó không đưa ra lời giải ngay mà khuyến khích người học tự suy nghĩ, thử nghiệm các quy tắc khác nhau, và quan sát kết quả. Quá trình này giúp củng cố sự hiểu biết về môi trường và các nguyên tắc điều khiển cơ bản. Một người học sẽ phải tự đặt ra các câu hỏi như:
    - Khi nào nên bật động cơ trái? (Khi tàu nghiêng sang phải, hoặc khi cần di chuyển sang phải).
    - Ngưỡng cho góc nghiêng nên là bao nhiêu? (0.1 radian? 0.2 radian?).
    - Làm thế nào để kết hợp nhiều quy tắc với nhau? (Ưu tiên quy tắc nào trước? Chống rơi, giữ thăng bằng, hay điều hướng?).

Phần 9: Hiện thực Tác tử Phản xạ Nâng cao (Ô Code 7)

Đây là phần lời giải cho thử thách ở trên.

```python
def improved_rocket_agent_function(observation):
    """
    A better reflex-based agent that uses all available thrusters.

    Controls:
    - Main thruster: Slow down descent when falling too fast
    - Left/Right thrusters: Control horizontal position, angle, and angular
velocity
    """

    # Extract observation values
    x = observation[Obs.X.value]           # Horizontal position
    y = observation[Obs.Y.value]           # Vertical position
    vx = observation[Obs.VX.value]         # Horizontal velocity
    vy = observation[Obs.VY.value]         # Vertical velocity
    angle = observation[Obs.ANGLE.value]   # Angle (0 = upright)
    angular_vel = observation[Obs.ANGULAR_VELOCITY.value]  # Angular velocity

    # Priority 1: Control descent speed with main thruster
    # Fire main thruster if falling too fast or too close to ground with high
speed
    if vy < -0.4 or (y < 0.3 and vy < -0.2):
        return Act.MAIN.value

    # Priority 2: Control angle - keep lander upright
    # If tilted significantly or rotating fast, use thrusters to stabilize
    if angle > 0.2 or angular_vel > 0.3:  # Tilted right or rotating right
        return Act.LEFT.value  # Fire left thruster to rotate left
    elif angle < -0.2 or angular_vel < -0.3:  # Tilted left or rotating left
        return Act.RIGHT.value  # Fire right thruster to rotate right

    # Priority 3: Control horizontal position - move toward center (x=0)
    # If moving away from center or too far from center, correct
    if x > 0.1 and vx > -0.1:  # Too far right or not moving left fast enough
        return Act.LEFT.value   # Fire left thruster to push right
    elif x < -0.1 and vx < 0.1:  # Too far left or not moving right fast enough
        return Act.RIGHT.value  # Fire right thruster to push left

    # Priority 4: Control horizontal velocity - slow down horizontal movement
    if abs(vx) > 0.3:  # Moving too fast horizontally
        if vx > 0:  # Moving right too fast
            return Act.LEFT.value   # Fire left thruster to slow down
        else:  # Moving left too fast
            return Act.RIGHT.value  # Fire right thruster to slow down
```

```
    # If everything looks good, do nothing
    return Act.NO_OP.value

# Test the improved agent
print("Testing improved agent with single episode:")
run_episode(improved_rocket_agent_function)
```

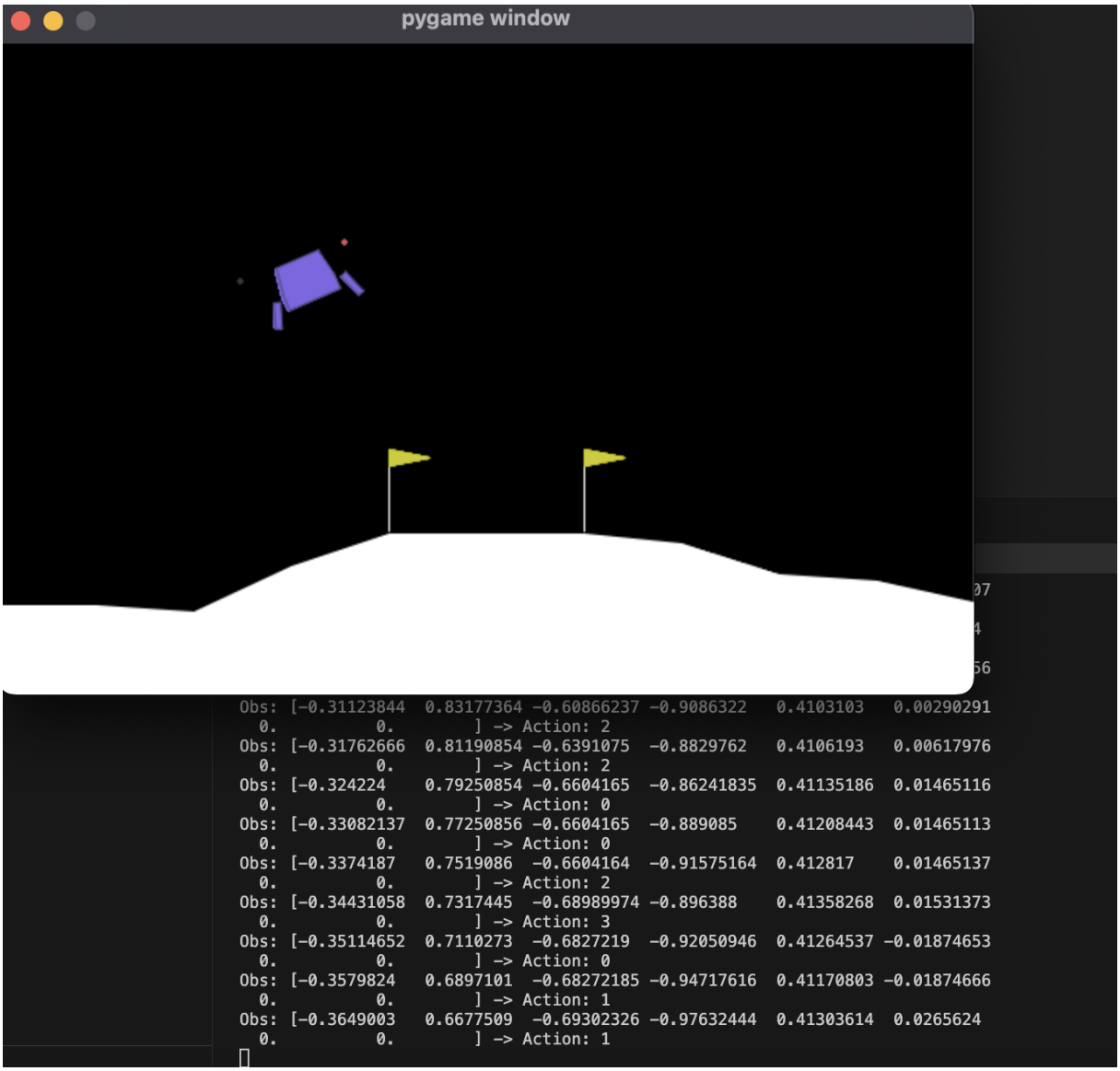
```
# Evaluate the improved agent performance
print("Evaluating improved agent over 100 episodes...")
improved_rewards = run_episodes(improved_rocket_agent_function, n=100)

print(f"Improved agent results:")
print(f"Average reward: {np.average(improved_rewards)}")
print(f"Success rate: {np.sum(np.array(improved_rewards) ==
100)}/{len(improved_rewards)}")
print(f"Success percentage: {np.sum(np.array(improved_rewards) ==
100)/len(improved_rewards)*100:.1f}%")

# Compare with original agent
print(f"\nComparison:")
print(f"Original agent success rate: 8/1000 (0.8%)")
print(f"Improved agent success rate: {np.sum(np.array(improved_rewards) ==
100)}/{len(improved_rewards)} ({np.sum(np.array(improved_rewards) ==
100)/len(improved_rewards)*100:.1f}%)")
print(f"Improvement factor: {(np.sum(np.array(improved_rewards) ==
100)/len(improved_rewards))/(8/1000):.1f}x")
```

```
Evaluating improved agent over 100 episodes...
Improved agent results:
Average reward: -100.0
Success rate: 0/100
Success percentage: 0.0%

Comparison:
Original agent success rate: 8/1000 (0.8%)
Improved agent success rate: 0/100 (0.0%)
Improvement factor: 0.0x
```

- **Phân tích sâu về logic:** Tác tử này được cấu trúc theo một hệ thống **ưu tiên**. Các quy tắc được kiểm tra từ trên xuống dưới, và ngay khi một điều kiện được thỏa mãn, một hành động sẽ được trả về và hàm kết thúc. Đây là một cách phổ biến để xử lý các quy tắc có thể xung đột.
    - **Ưu tiên 1: Chống rơi (quan trọng nhất).**

- if vy < -0.4: Nếu tàu rơi quá nhanh (nhanh hơn ngưỡng -0.3 của tác tử trước), bật động cơ chính.
- or (y < 0.3 and vy < -0.2): Một quy tắc an toàn bổ sung. Nếu tàu đã ở rất gần mặt đất (y < 0.3) và vẫn đang rơi (vy < -0.2), hãy bật động cơ chính ngay cả khi tốc độ rơi chưa quá lớn. Điều này giúp "phanh gấp" trước khi va chạm.
- **Tại sao đây là ưu tiên 1?** Vì nếu tàu rơi, tất cả các nỗ lực khác đều vô nghĩa. Sống sót là ưu tiên hàng đầu.

- **Ưu tiên 2: Giữ thăng bằng.**
  - if angle > 0.2 or angular_vel > 0.3: Nếu tàu nghiêng sang phải (angle > 0.2) hoặc đang quay sang phải (angular_vel > 0.3), bật động cơ **trái**. Động cơ trái sẽ tạo mô-men xoay ngược chiều kim đồng hồ, giúp tàu đứng thẳng lại.
  - elif angle < -0.2 or angular_vel < -0.3: Ngược lại, nếu tàu nghiêng/quay sang trái, bật động cơ **phải**.
  - **Tại sao đây là ưu tiên 2?** Vì nếu tàu không thẳng đứng, động cơ chính sẽ không hoạt động hiệu quả và việc hạ cánh sẽ thất bại.
- **Ưu tiên 3: Điều hướng về bãi đáp.**
  - if x > 0.1 and vx > -0.1: Nếu tàu ở bên phải bãi đáp (x > 0.1) và không di chuyển đủ nhanh sang trái (vx > -0.1), bật động cơ **trái**. Động cơ trái tạo lực đẩy sang phải, nhưng do góc nghiêng, nó cũng có thể được dùng để điều hướng. Logic ở đây có vẻ hơi phản trực giác. Một cách tiếp cận tốt hơn có thể là: nghiêng tàu một chút rồi bật động cơ chính. Tuy nhiên, tác giả đang cố gắng tạo ra các quy tắc đơn giản.
  - elif x < -0.1 and vx < 0.1: Tương tự cho phía bên trái.
  - **Tại sao đây là ưu tiên 3?** Sau khi đã đảm bảo không rơi và thăng bằng, mục tiêu tiếp theo là đi đến đúng vị trí.
- **Ưu tiên 4: Giảm tốc độ ngang.**
  - if abs(vx) > 0.3: Nếu tàu bay ngang quá nhanh, cần giảm tốc.
  - if vx > 0: Nếu bay sang phải, bật động cơ **trái** để tạo lực cản.
  - else: Nếu bay sang trái, bật động cơ **phải**.


pygame window
Obs: [-0.31123844 0.83177364 -0.60866237 -0.9086322   0.4103103   0.00290291
  0.   0.   ] -> Action: 2
Obs: [-0.31762666 0.81190854 -0.6391075 -0.8829762   0.4106193   0.00617976
  0.   0.   ] -> Action: 2
Obs: [-0.324224   0.79250854 -0.6604165 -0.86241835 0.41135186 0.01465116
  0.   0.   ] -> Action: 0
Obs: [-0.33082137 0.77250856 -0.6604165 -0.889085   0.41208443 0.01465113
  0.   0.   ] -> Action: 0
Obs: [-0.3374187   0.7519086 -0.6604164 -0.91575164 0.412817   0.01465137
  0.   0.   ] -> Action: 2
Obs: [-0.34431058 0.7317445 -0.68989974 -0.896388   0.41358268 0.01531373
  0.   0.   ] -> Action: 3
Obs: [-0.35114652 0.7110273 -0.6827219 -0.92050946 0.41264537 -0.01874653
  0.   0.   ] -> Action: 0
Obs: [-0.3579824   0.6897101 -0.68272185 -0.94717616 0.41170803 -0.01874666
  0.   0.   ] -> Action: 1
Obs: [-0.3649003   0.6677509 -0.69302326 -0.97632444 0.41303614 0.0265624
  0.   0.   ] -> Action: 1

CHƯƠNG 2: GIẢI QUYẾT BÀI TOÁN BẰNG TÌM KIẾM (SEARCH)

1. Giới thiệu Chương 3

Chương 3 tập trung vào một trong những chủ đề nền tảng và có sức ảnh hưởng nhất của Trí tuệ Nhân tạo: **Giải quyết bài toán bằng tìm kiếm (Problem Solving by Search)**. Trong thực tế, rất nhiều bài toán từ đơn giản đến phức tạp có thể được mô hình hóa thành bài toán tìm kiếm một lời giải trong một không gian các khả năng. Các tác nhân (agents) thông minh, từ robot tự hành, hệ thống định vị GPS cho đến các chương trình chơi game, đều phải đối mặt với các vấn đề cần được giải quyết thông qua một chuỗi các hành động. Phương pháp tìm kiếm cung cấp một khuôn khổ tổng quát, mạnh mẽ và có hệ thống để giải quyết các vấn đề này.

Để chính thức hóa một bài toán cho thuật toán tìm kiếm, cần định nghĩa các thành phần chính sau:

- Không gian trạng thái (State Space): Đây là một tập hợp trừu tượng chứa tất cả các trạng thái mà môi trường có thể đạt được. Ví dụ, trong bài toán cờ vua, không gian trạng thái bao gồm mọi cách sắp xếp các quân cờ hợp lệ trên bàn cờ.
- Trạng thái ban đầu (Initial State): Là trạng thái xuất phát của tác nhân, điểm khởi đầu của quá trình tìm kiếm.
- Hành động (Actions): Là tập hợp các hành động mà tác nhân có thể thực hiện. Trong một trạng thái cụ thể, tác nhân sẽ có một danh sách các hành động khả thi.
- Hàm chuyển đổi (Transition Model): Mô tả kết quả của việc thực hiện một hành động. Nó nhận đầu vào là một trạng thái và một hành động, và trả về trạng thái kết quả. Ví dụ, Result(s, a) = s'.
- Phép thử mục tiêu (Goal Test): Là một hàm xác định xem một trạng thái đã cho có phải là trạng thái mục tiêu hay không. Có thể có một hoặc nhiều trạng thái mục tiêu.
- Chi phí đường đi (Path Cost): Gán một chi phí số cho mỗi đường đi. Chi phí này thường là tổng chi phí của các hành động dọc theo đường đi đó. Mục tiêu thường là tìm ra đường đi có chi phí thấp nhất.

Các thuật toán tìm kiếm được chia thành hai loại chính, mỗi loại có ưu và nhược điểm riêng:

1. Tìm kiếm mù: Các thuật toán này không có thông tin bổ sung về bài toán ngoài định nghĩa của nó. Chúng chỉ biết cách duyệt qua không gian trạng thái một cách có hệ thống. Ví dụ:
    - Tìm kiếm theo chiều rộng (BFS): Mở rộng các nút nông nhất trước tiên, đảm bảo tìm ra lời giải tối ưu nếu chi phí mỗi bước là như nhau.
    - Tìm kiếm theo chiều sâu (DFS): Mở rộng các nút sâu nhất trước tiên, có thể tìm ra lời giải nhanh nhưng không đảm bảo tối ưu.
    - Tìm kiếm chi phí đồng nhất (UCS): Mở rộng các nút có chi phí đường đi thấp nhất, là một phiên bản tổng quát của BFS.
2. Tìm kiếm có thông tin: Các thuật toán này sử dụng kiến thức đặc thù của bài toán dưới dạng một hàm heuristic h(n) để ước tính chi phí từ nút n đến mục tiêu. Heuristic giúp dẫn dắt quá trình tìm kiếm đến những hướng hứa hẹn hơn, giúp tìm ra lời giải nhanh hơn đáng kể. Ví dụ:
    - Tìm kiếm Tham lam (Greedy Best-First Search): Mở rộng nút được đánh giá là gần mục tiêu nhất dựa trên heuristic.
    - A*: Kết hợp cả chi phí đường đi đã qua g(n) và chi phí ước tính đến mục tiêu h(n). Đây là một trong những thuật toán tìm kiếm quan trọng và hiệu quả nhất.

Trong bài thực hành này, sẽ áp dụng các thuật toán tìm kiếm này để giải quyết bài toán kinh điển: tìm đường đi trong mê cung.

2. Nội dung chi tiết và Kết quả thực nghiệm

2.1. Bài toán Mê cung và Tác nhân dựa trên mục tiêu (Maze.ipynb)

Mục tiêu cốt lõi là xây dựng một tác nhân thông minh có khả năng tự động tìm ra đường đi từ điểm bắt đầu (S) đến điểm kết thúc (G) trong một môi trường mê cung tĩnh, đã biết trước.

Cấu trúc và Triển khai

- Biểu diễn Mê cung: Mê cung được đọc từ file văn bản (.txt) và được chuyển đổi thành một mảng NumPy 2D. Mỗi phần tử trong mảng đại diện cho một ô trong

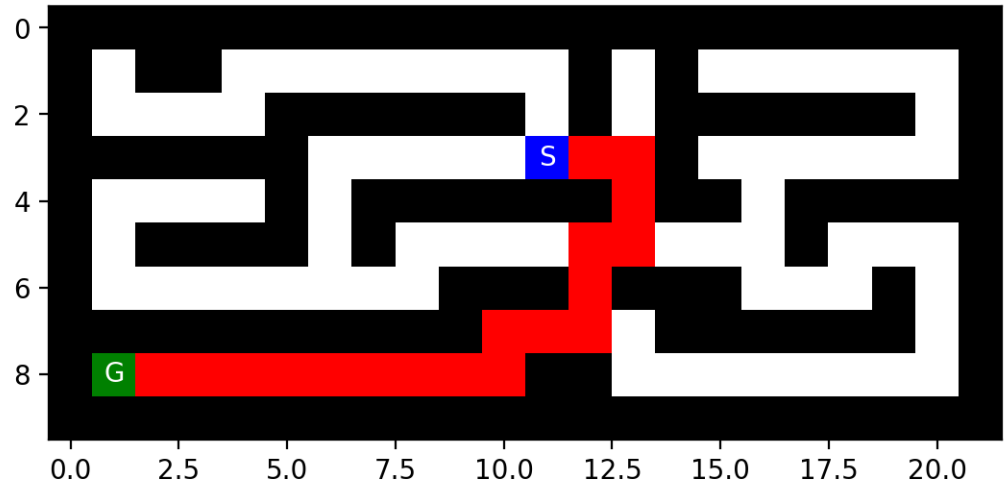
mê cung, với các ký tự quy ước: '#' cho tường, ' ' cho đường đi, 'S' cho điểm bắt đầu, và 'G' cho điểm kết thúc. Cách biểu diễn này rất trực quan và hiệu quả cho việc truy xuất và kiểm tra trạng thái của từng ô.

- Cấu trúc Node: Để theo dõi quá trình tìm kiếm, một class Node được định nghĩa. Đây không chỉ là một trạng thái (tọa độ) mà là một cấu trúc dữ liệu hoàn chỉnh chứa thông tin cần thiết để tái tạo lại đường đi và tính toán chi phí. Nó bao gồm:
    - state: Tọa độ (hàng, cột) của ô hiện tại.
    - parent: Tham chiếu đến Node cha đã tạo ra node này. Đây là thành phần quan trọng để có thể truy ngược lại đường đi từ đích về điểm xuất phát.
    - action: Hành động ('U', 'D', 'L', 'R') đã được thực hiện để đi từ parent đến state hiện tại.
    - path_cost: Tổng chi phí để đi từ trạng thái ban đầu đến node này, hay còn gọi là g(n).
- Hàm tìm kiếm chính best_first_search: Đây là một hàm tìm kiếm tổng quát, được thiết kế để có thể triển khai nhiều thuật toán khác nhau. Sự khác biệt giữa các thuật toán nằm ở cách tính toán "giá trị ưu tiên" để sắp xếp các node trong frontier (hàng đợi ưu tiên). Hàm này sử dụng hai cấu trúc dữ liệu chính:
    - Frontier: Một hàng đợi ưu tiên (priority queue) chứa các node đã được phát hiện nhưng chưa được khám phá. Node có giá trị ưu tiên thấp nhất sẽ được lấy ra để khám phá tiếp theo.
    - Reached (Explored Set): Một tập hợp (set) chứa các trạng thái đã được khám phá để tránh việc xử lý lặp lại một trạng thái nhiều lần, ngăn chặn các vòng lặp vô tận.

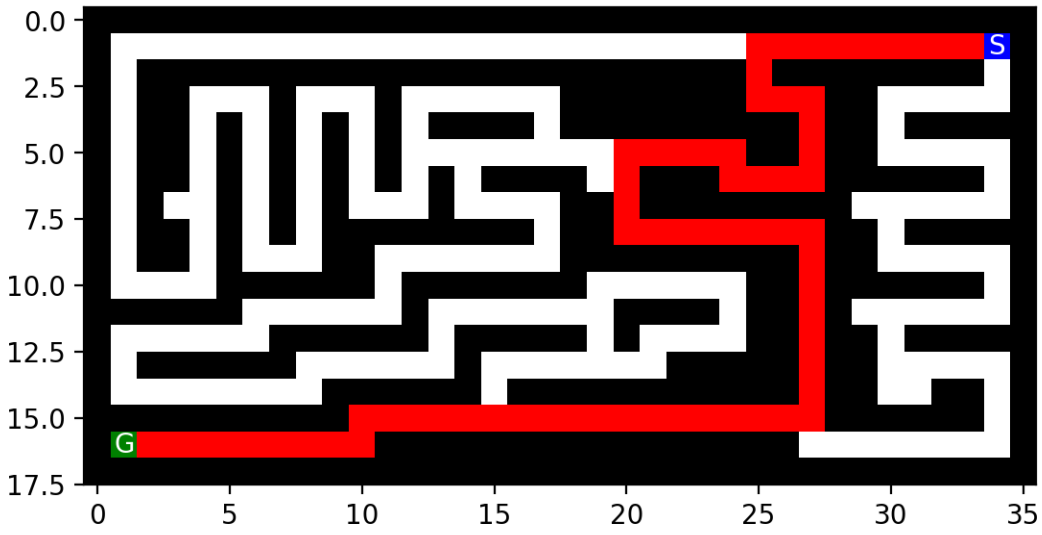
Cách triển khai các thuật toán cụ thể:

- BFS: Ưu tiên node dựa trên path_cost. Vì mỗi bước đi có chi phí là 1, điều này tương đương với việc ưu tiên các node ở độ sâu nông hơn trước.

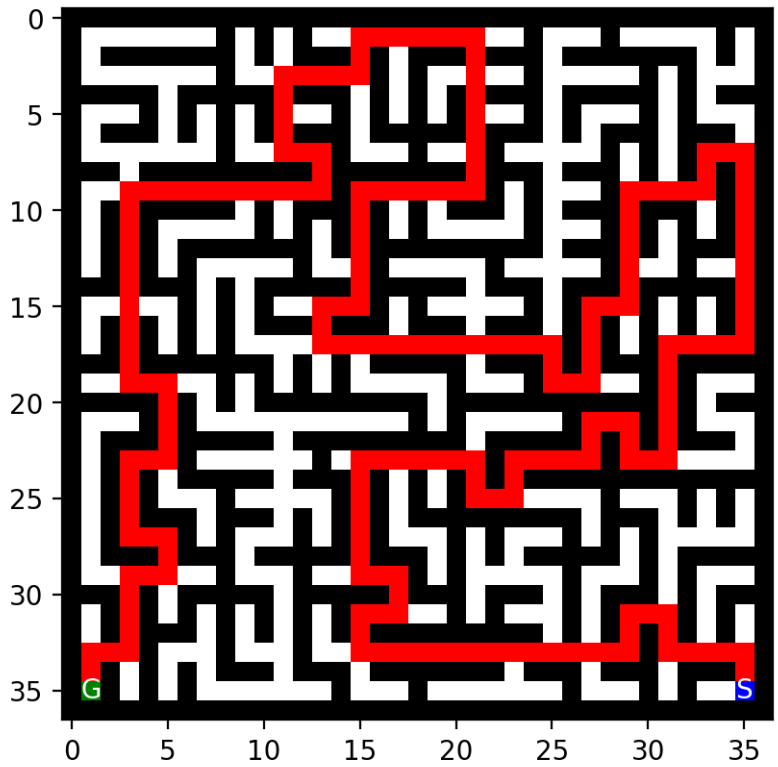
small_maze.txt



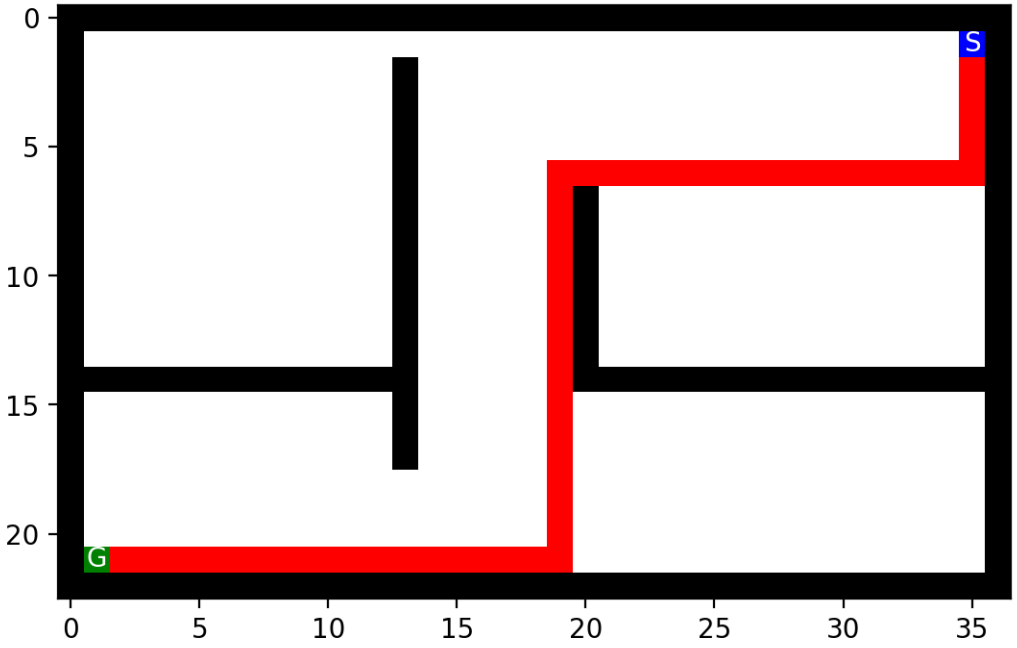
medium_maze.txt



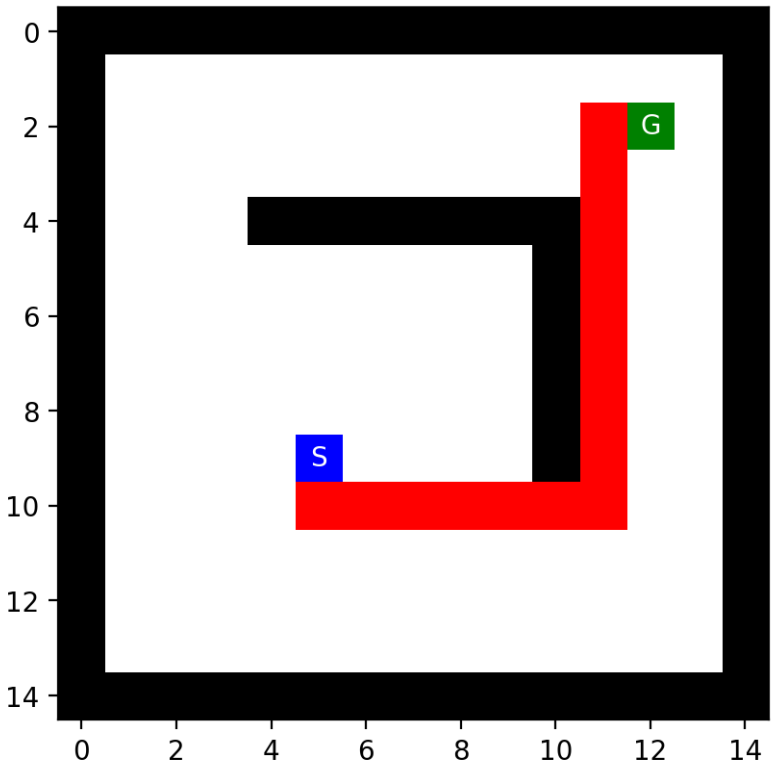
large_maze.txt



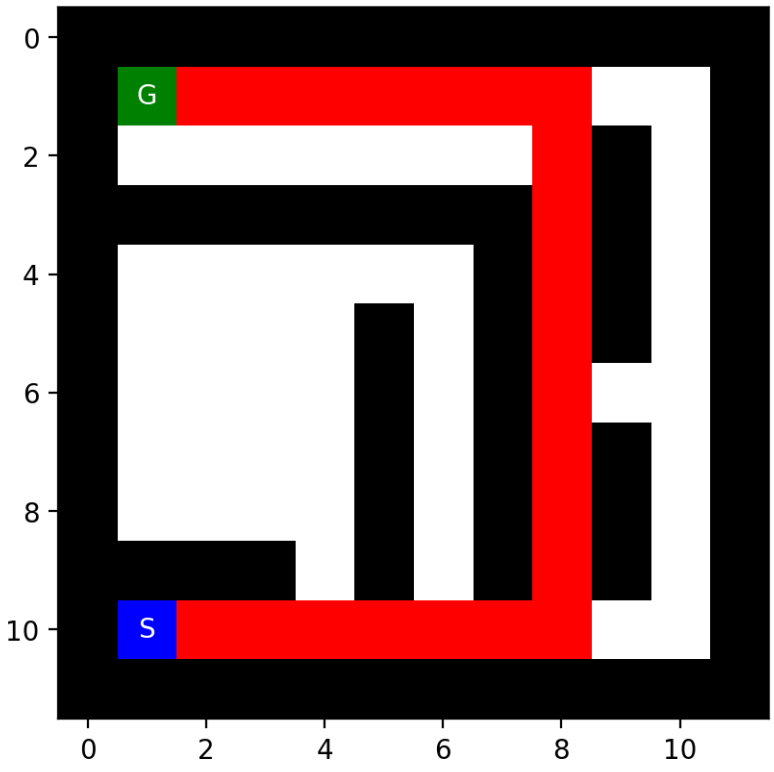
open_maze.txt



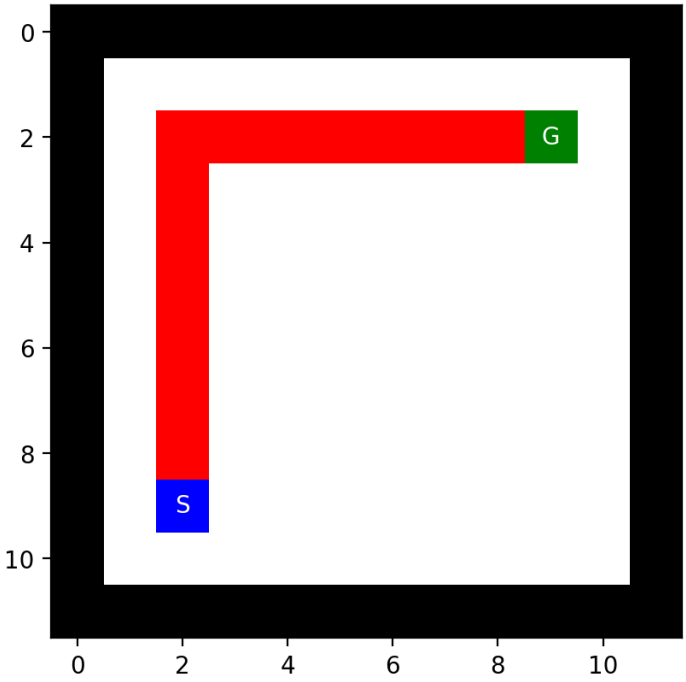
wall_maze.txt



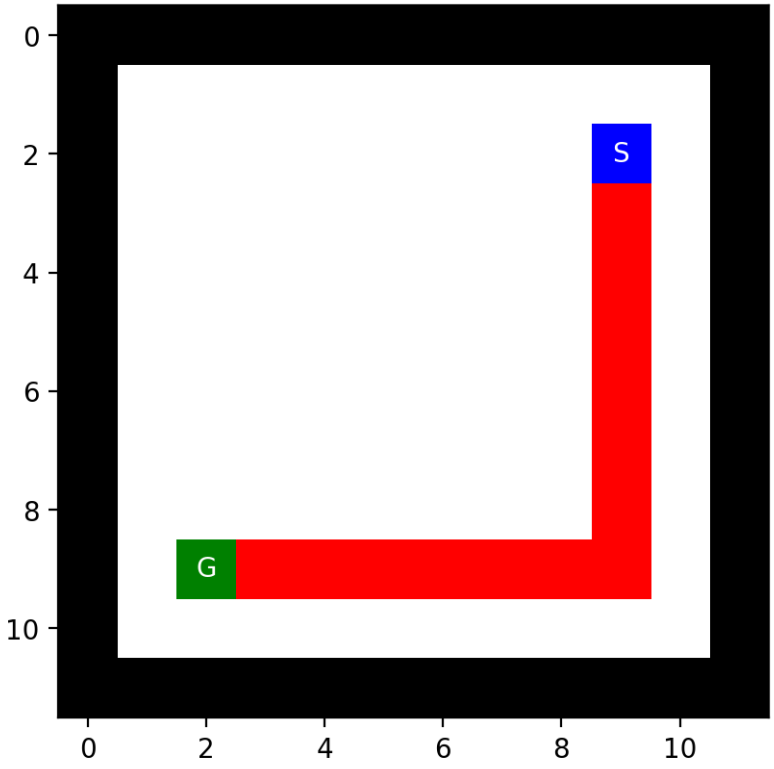
loops_maze.txt



empty_maze_2.txt

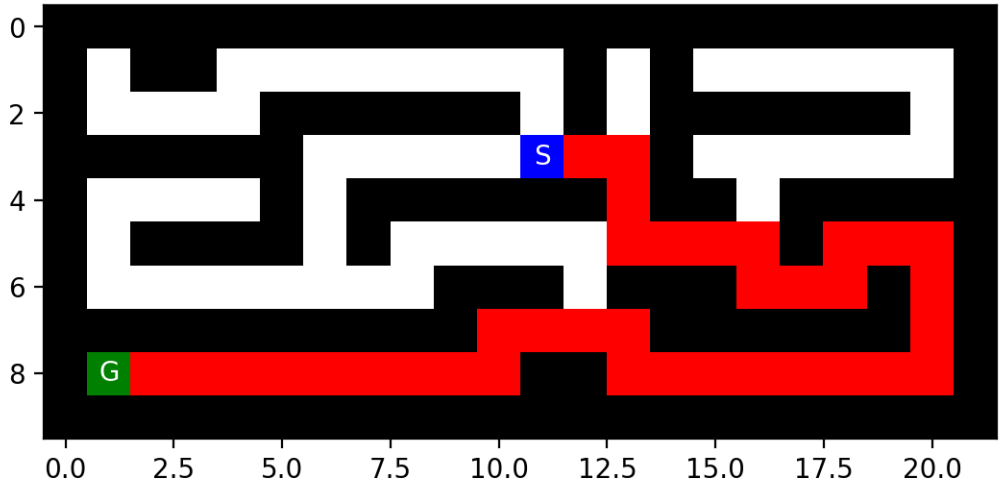


empty_maze.txt

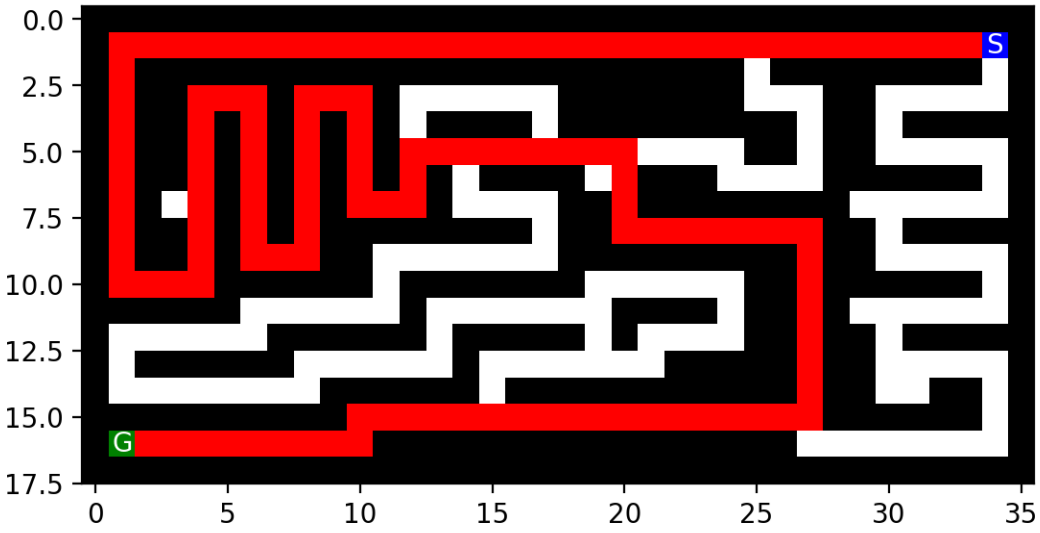


- **DFS:** Ưu tiên node dựa trên path_cost âm (hoặc thứ tự ngược lại), tương đương với việc đi sâu nhất có thể vào một nhánh. (Trường hợp Open_maze thì không tìm thấy lời giải (timeout))

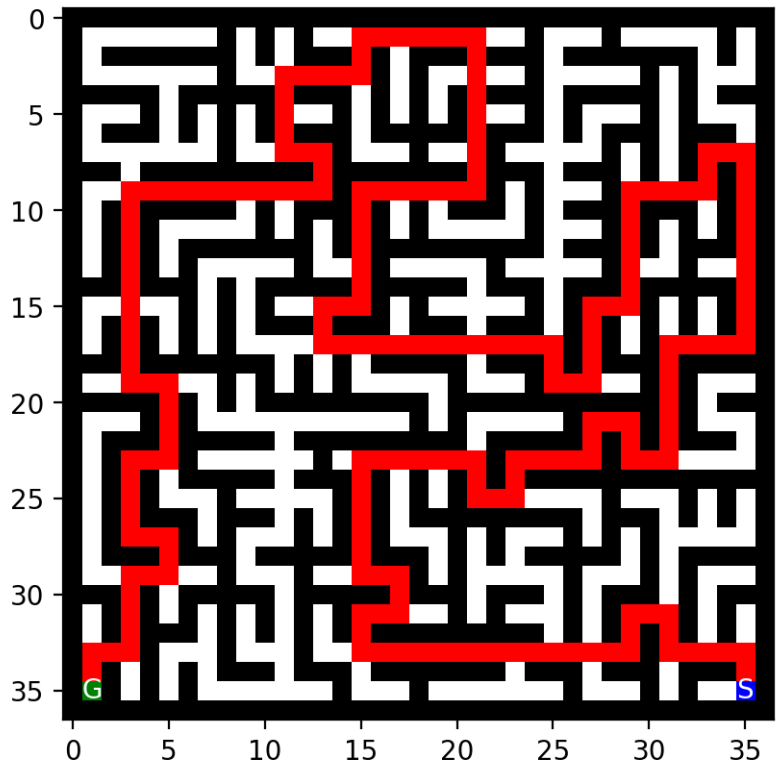
small_maze.txt



medium_maze.txt



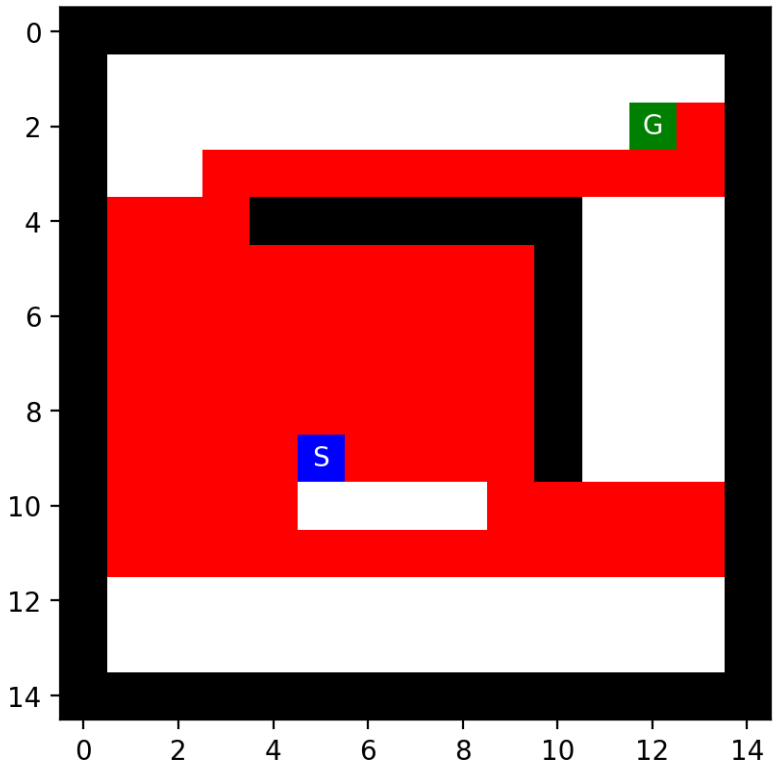
large_maze.txt



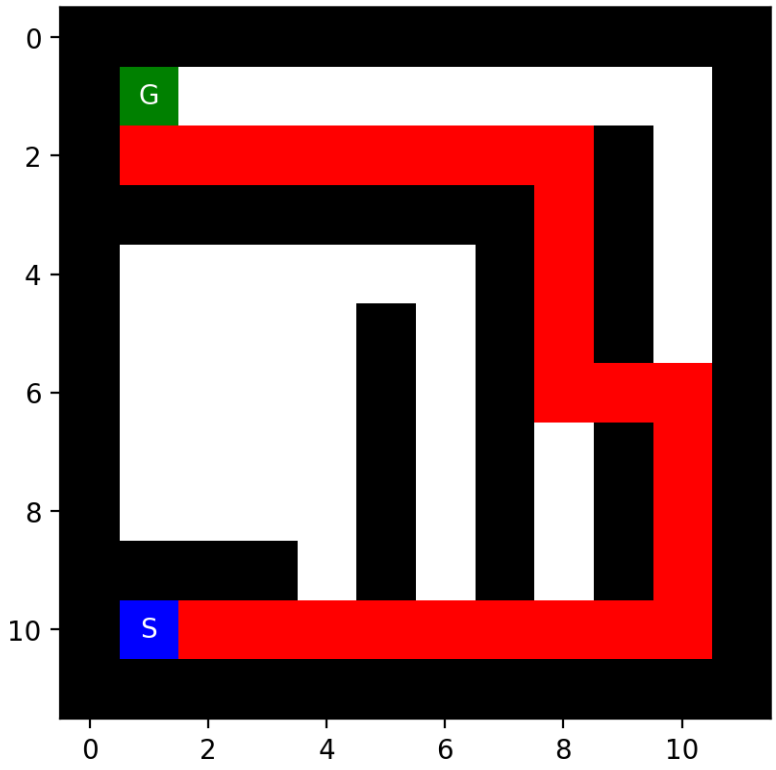
open_maze.txt

```
Solution for DFS: Không tìm thấy lời giải (timeout).
```

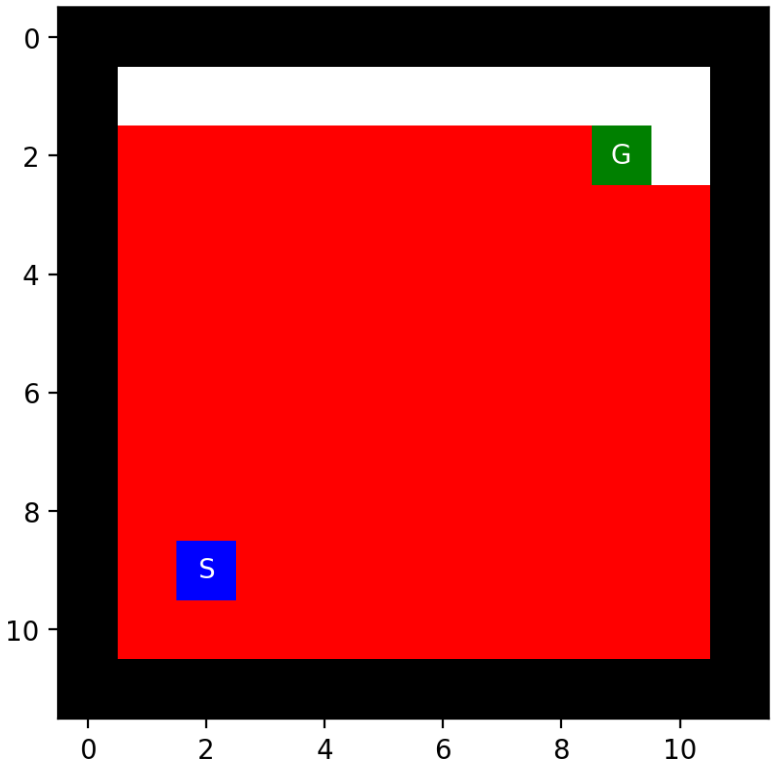
wall_maze.txt



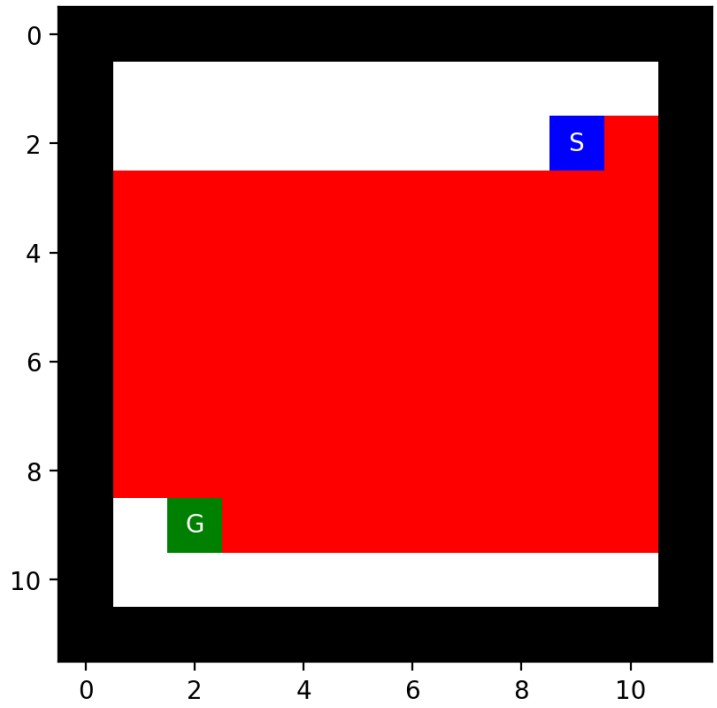
loops_maze.txt



empty_maze.txt

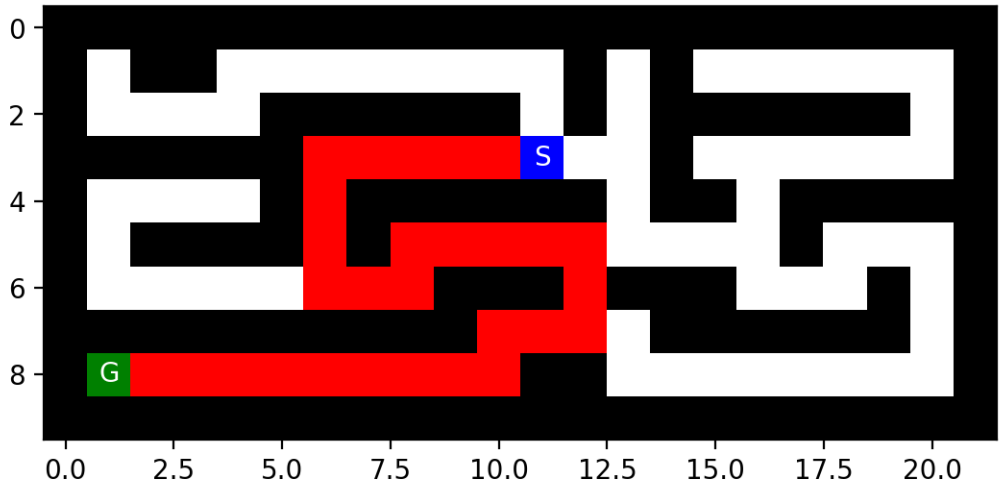


empty_maze_2.txt



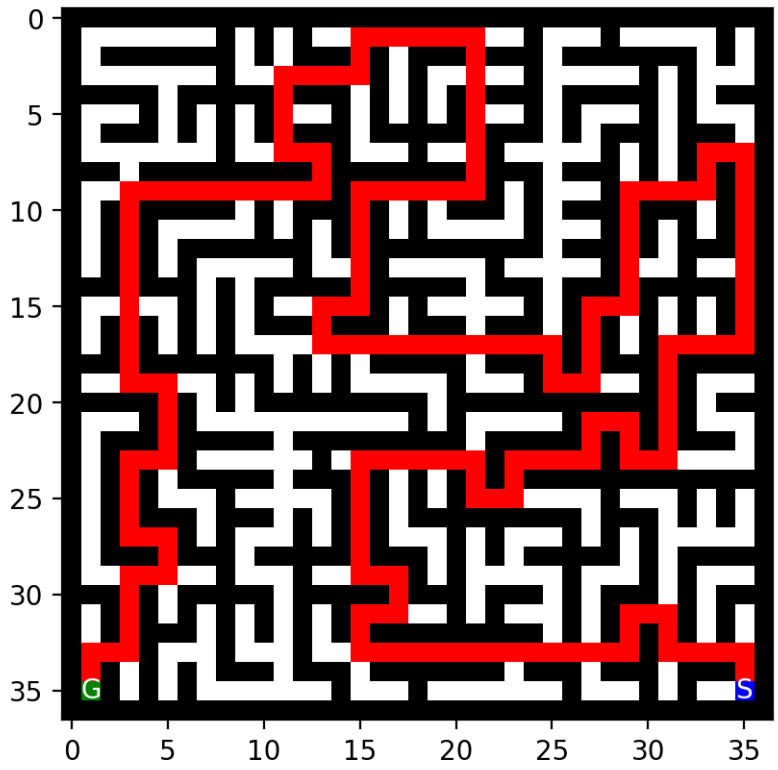
- GBFS: Ưu tiên node chỉ dựa trên giá trị heuristic h(n) (khoảng cách Manhattan từ node hiện tại đến đích).

small_maze.txt

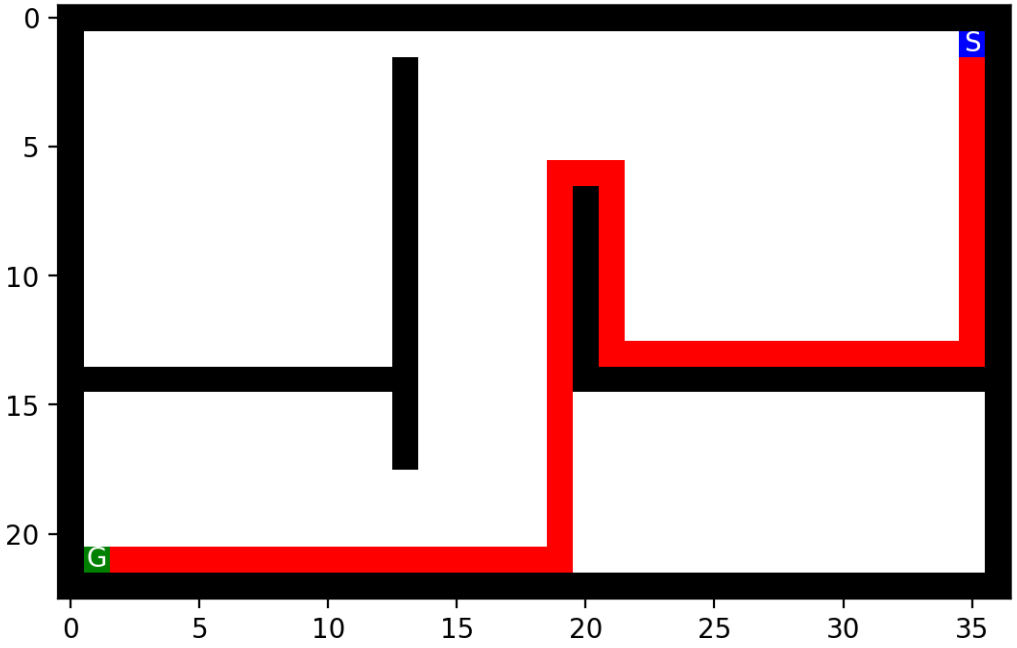


medium_maze.txt

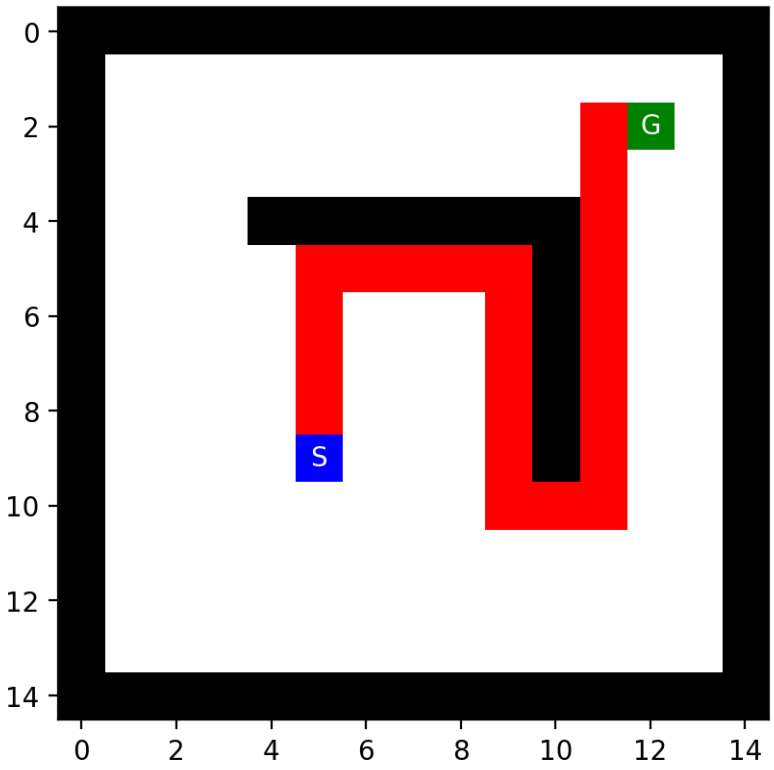
large_maze.txt



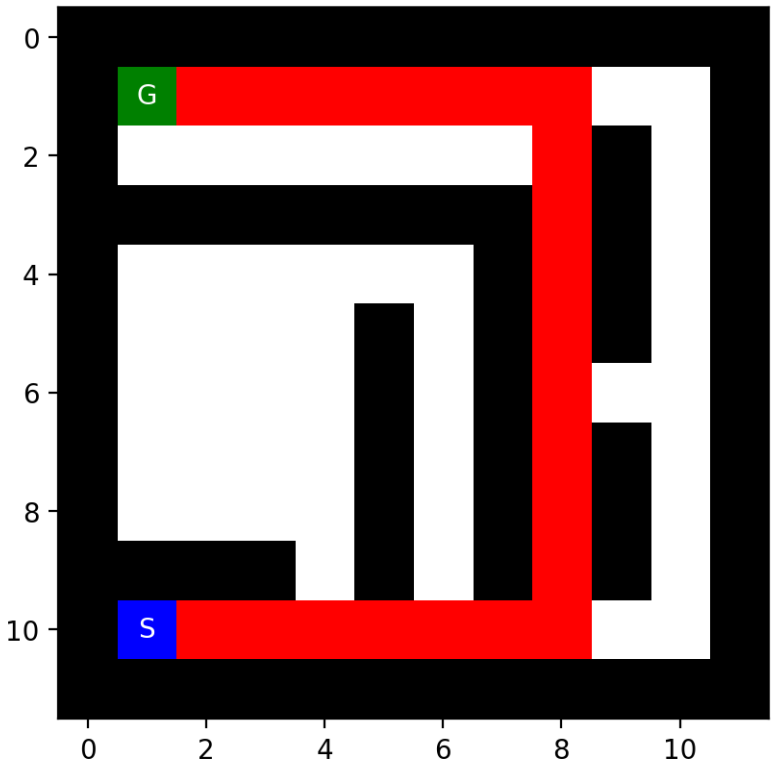
open_maze.txt



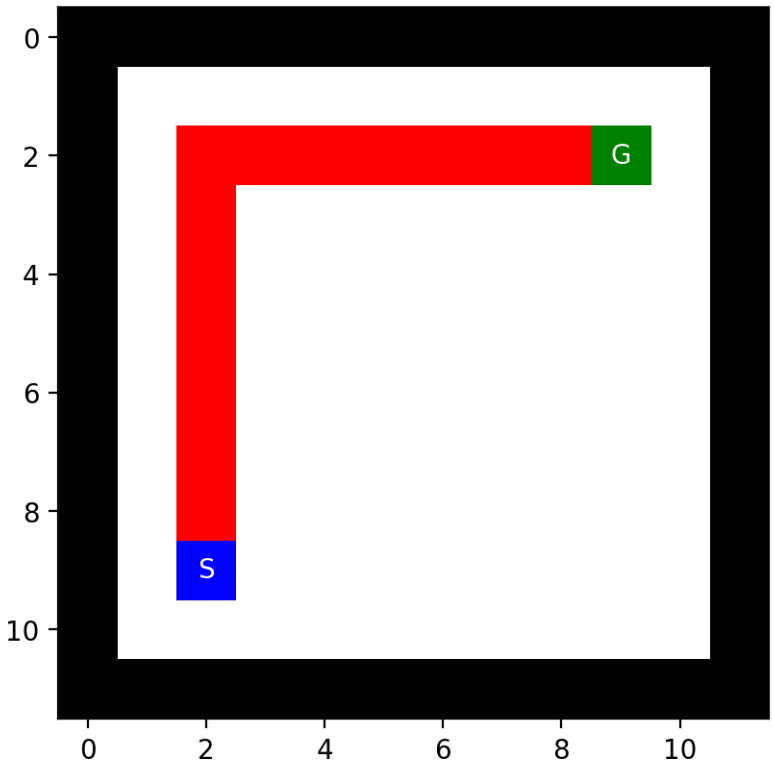
wall_maze.txt



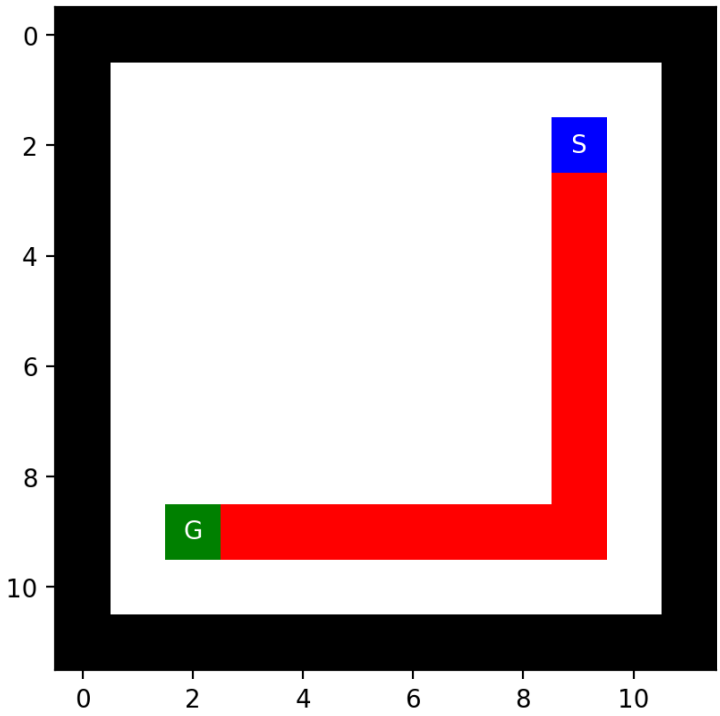
loops_maze.txt



empty_maze.txt

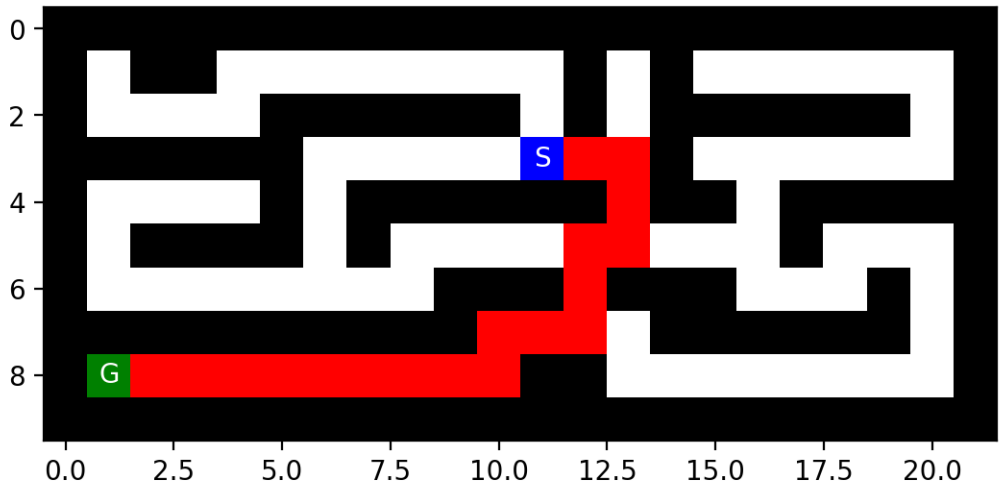


empty_maze_2.txt

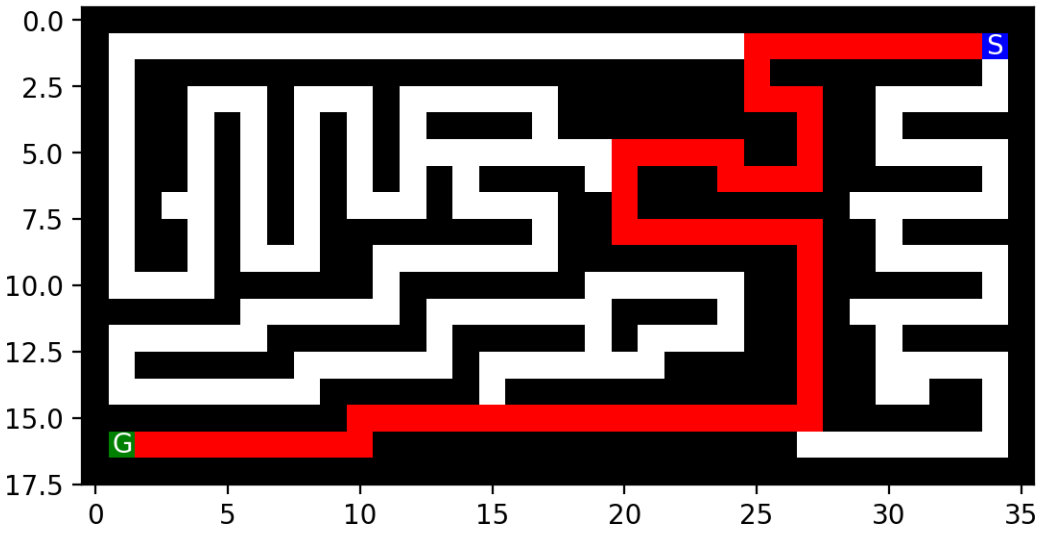


- A* Search: Ưu tiên node dựa trên tổng f(n) = g(n) + h(n), trong đó g(n) là path_cost và h(n) là heuristic.

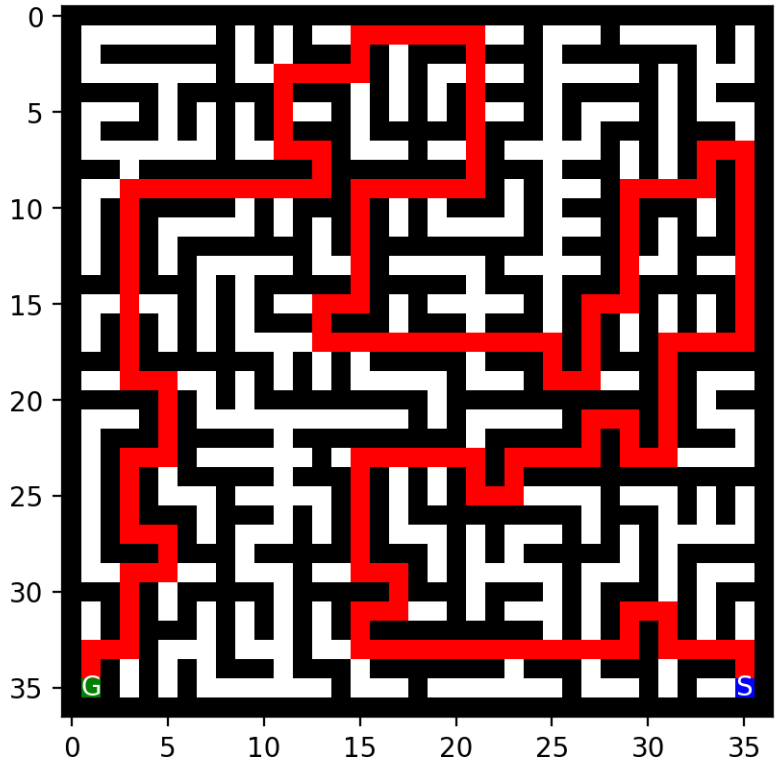
small_maze.txt



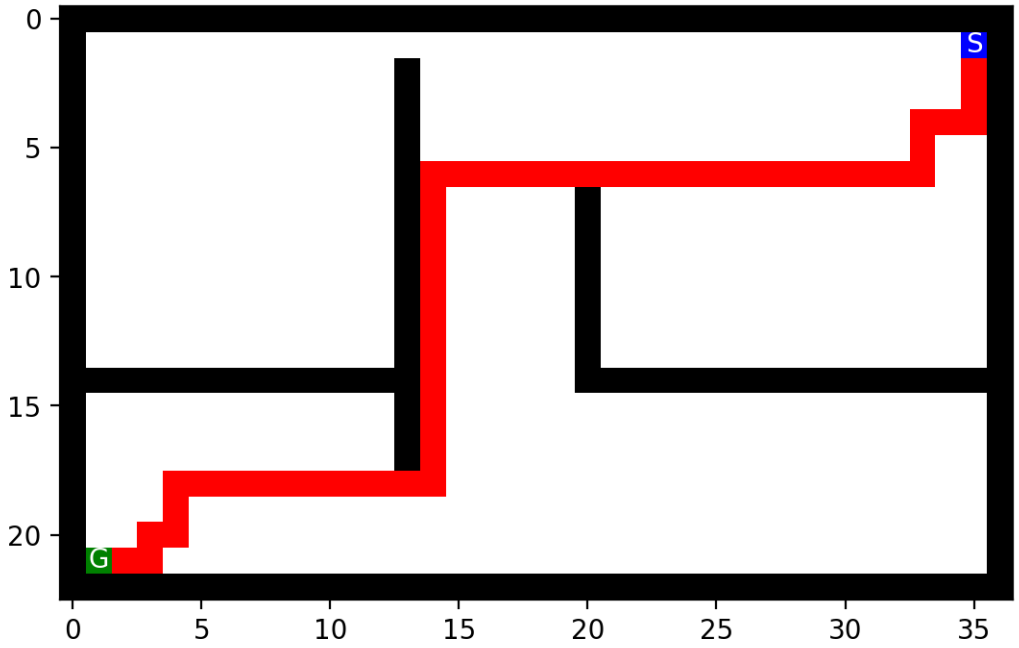
medium_maze.txt



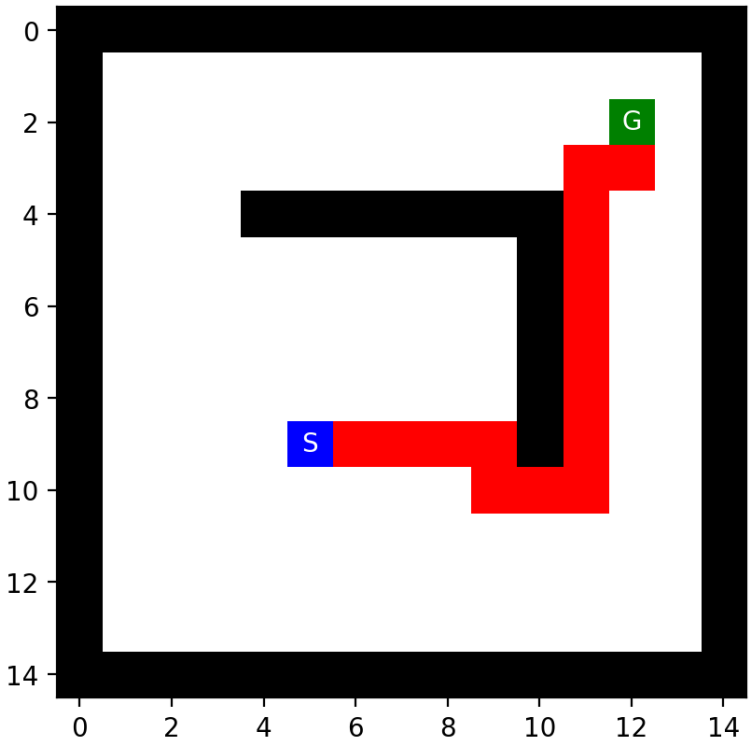
large_maze.txt



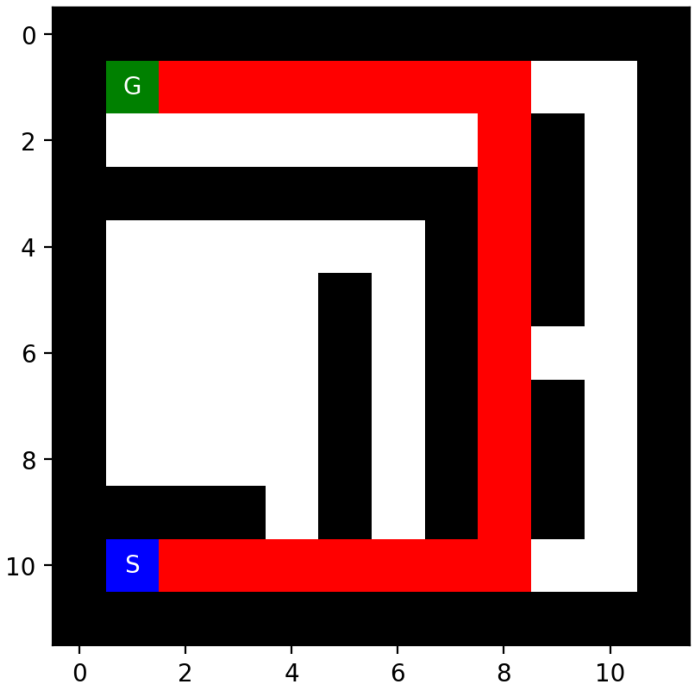
open_maze.txt



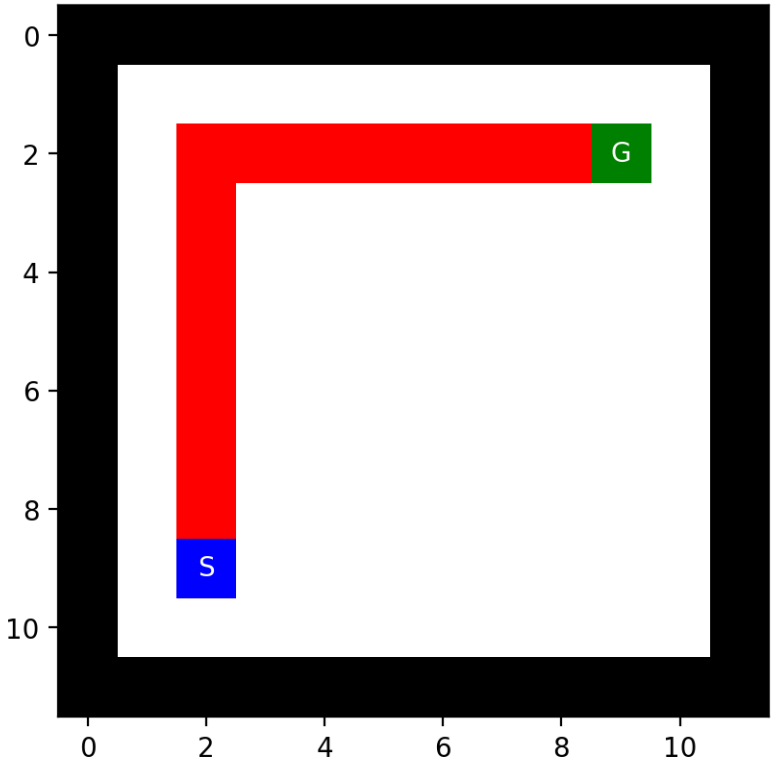
wall_maze.txt



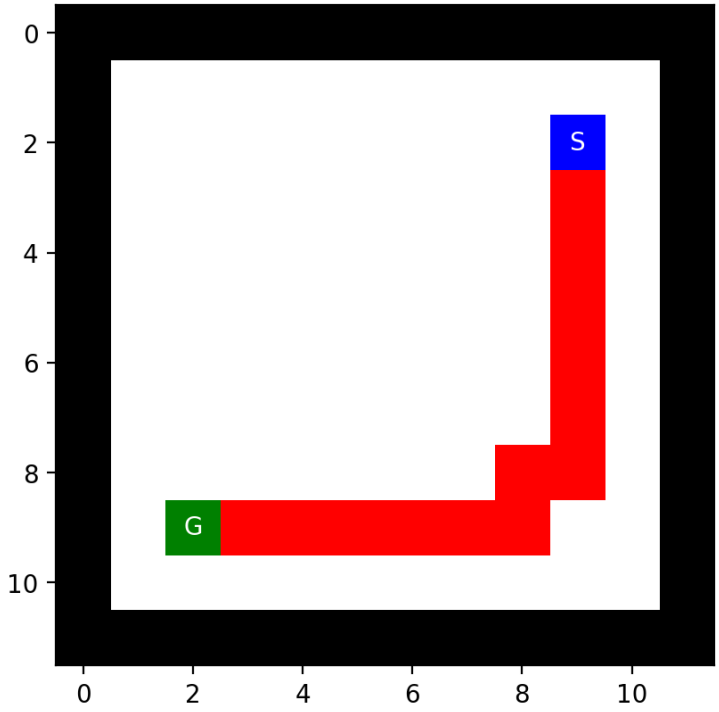
loops_maze.txt



empty_maze.txt



empty_maze_2.txt



2.2. Các mê cung sử dụng trong thực nghiệm (Show_all_mazes.ipynb)

Để đánh giá và so sánh hiệu quả của các thuật toán một cách toàn diện, việc thử nghiệm trên một bộ dữ liệu đa dạng là vô cùng quan trọng. File Show_all_mazes.ipynb được tạo ra để đọc và hiển thị tất cả các file maze_*.txt trong thư mục, giúp có một cái nhìn tổng quan về các thử thách mà tác nhân sẽ phải đối mặt.

Dưới đây là mô tả chi tiết hơn về ý nghĩa của từng loại mê cung:

- empty_maze.txt: Mê cung trống, không có vật cản. Đây là trường hợp cơ bản nhất để kiểm tra tính đúng đắn của thuật toán và là kịch bản mà heuristic thể hiện chính xác nhất chi phí thực tế.

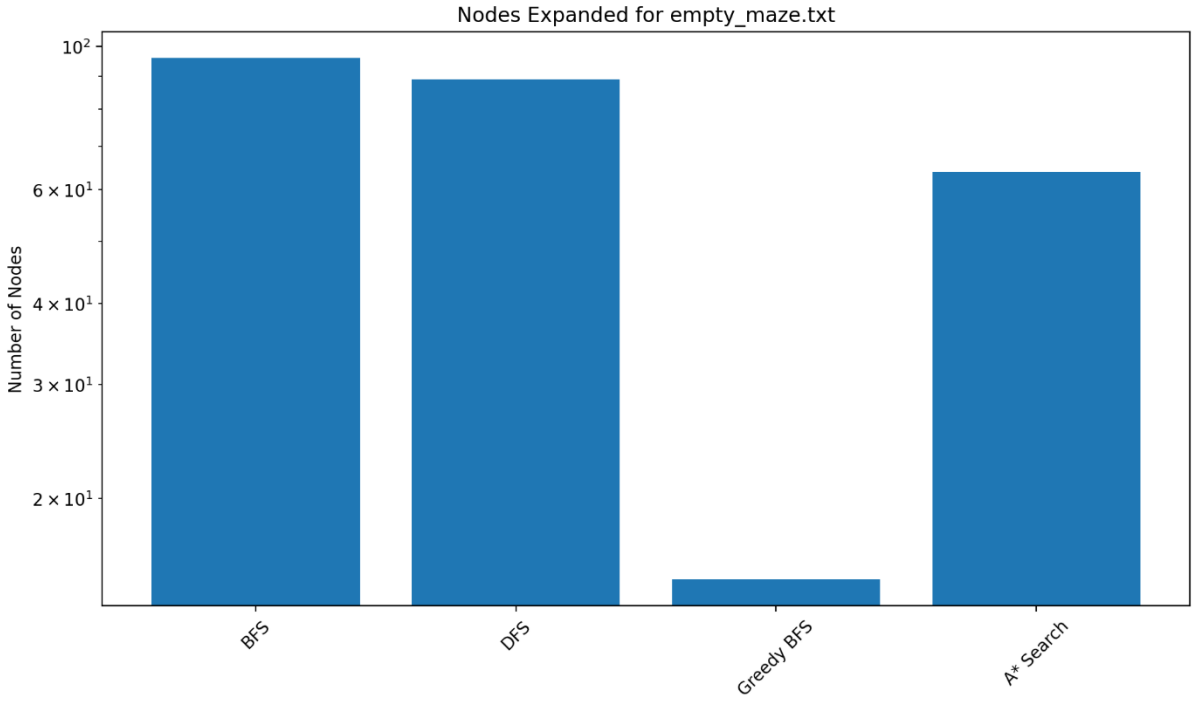
--- Results for empty_maze.txt ---

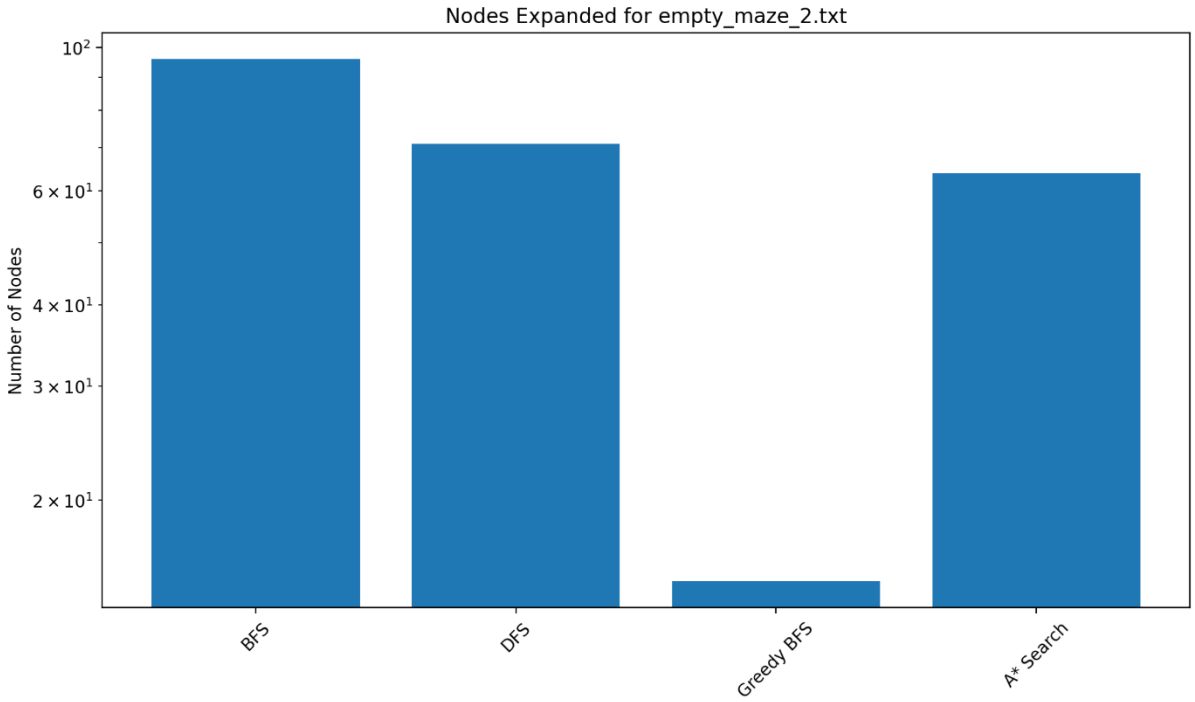
	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	14	96	14	12	100	0.0011s
1	DFS	88	89	88	81	169	0.0042s
2	Greedy BFS	14	15	14	29	71	0.0005s
3	A* Search	14	64	14	31	124	0.0012s

--- Results for empty_maze_2.txt ---

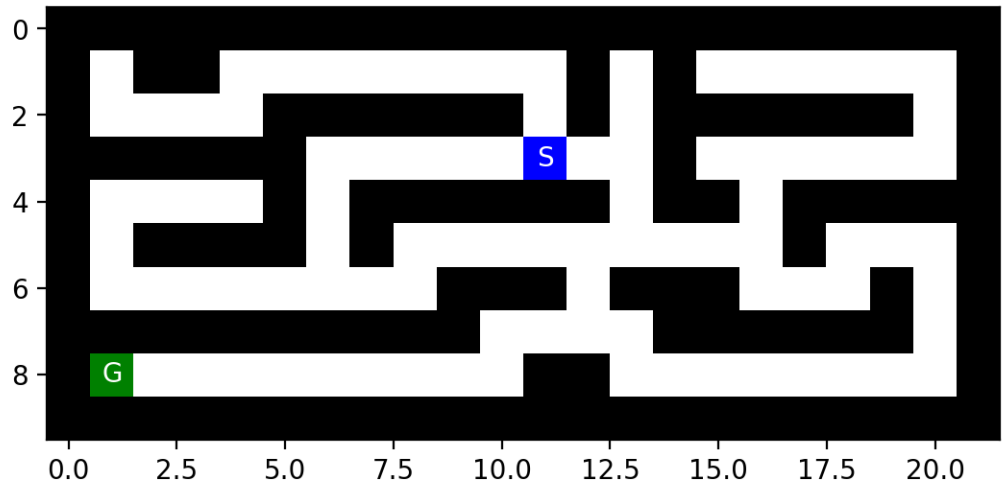
	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	14	96	14	12	100	0.0008s
1	DFS	70	71	70	75	145	0.0009s
2	Greedy BFS	14	15	14	29	71	0.0003s
3	A* Search	14	64	14	31	124	0.0007s

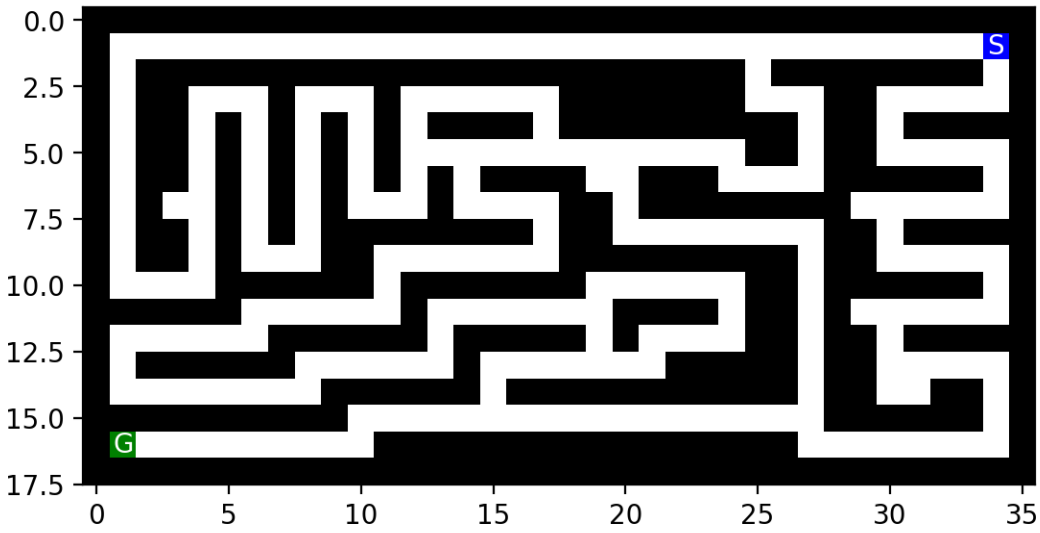
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.39 µs





- small_maze.txt, medium_maze.txt, large_maze.txt: Các mê cung có kích thước tăng dần. Chúng được sử dụng để đánh giá khả năng mở rộng (scalability) của các thuật toán. Khi không gian tìm kiếm tăng lên, sự khác biệt về hiệu năng giữa tìm kiếm mù và tìm kiếm có thông tin sẽ trở nên rõ rệt hơn.



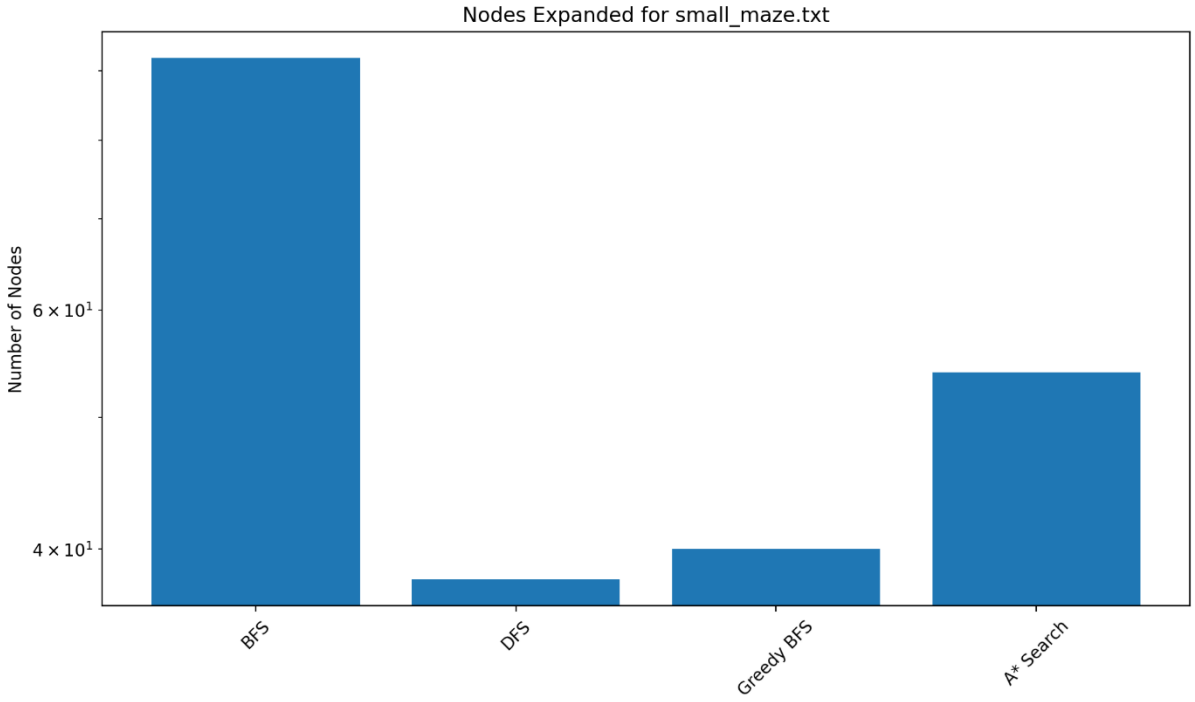


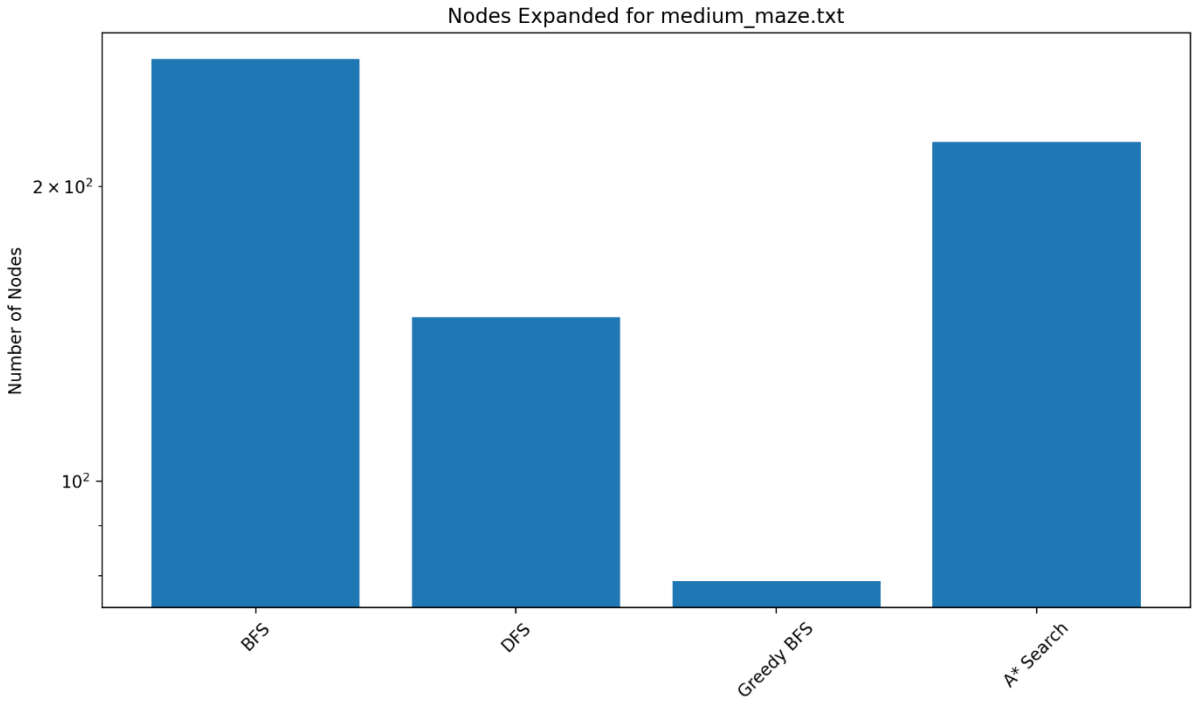
--- Results for small_maze.txt ---

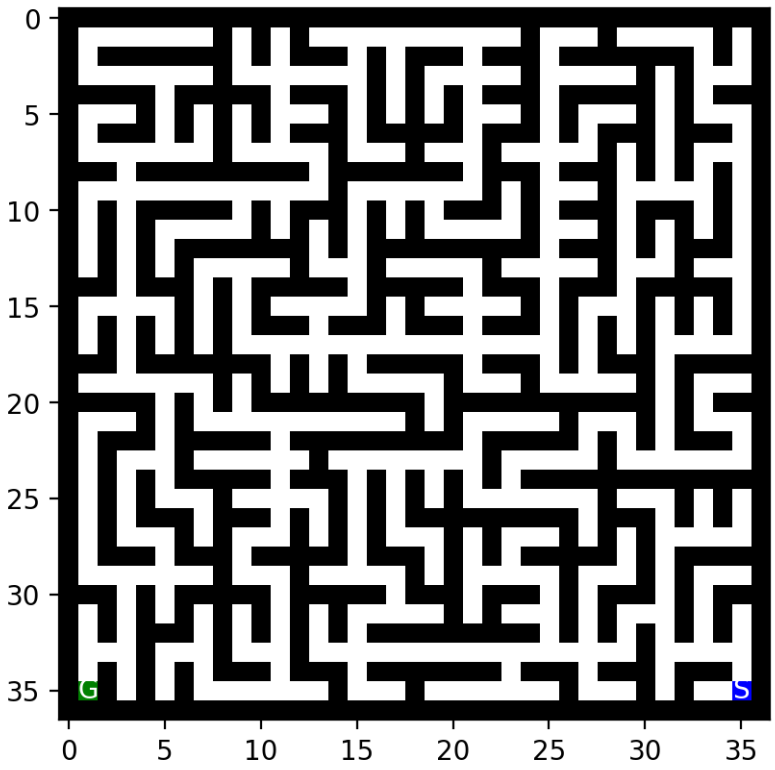
	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	19	92	19	8	94	0.0009s
1	DFS	37	38	37	7	44	0.0008s
2	Greedy BFS	29	40	29	5	48	0.0004s
3	A* Search	19	54	19	8	64	0.0005s

--- Results for medium_maze.txt ---

	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	68	270	68	8	274	0.0033s
1	DFS	130	147	130	9	139	0.0045s
2	Greedy BFS	74	79	74	4	85	0.0009s
3	A* Search	68	222	68	8	234	0.0055s

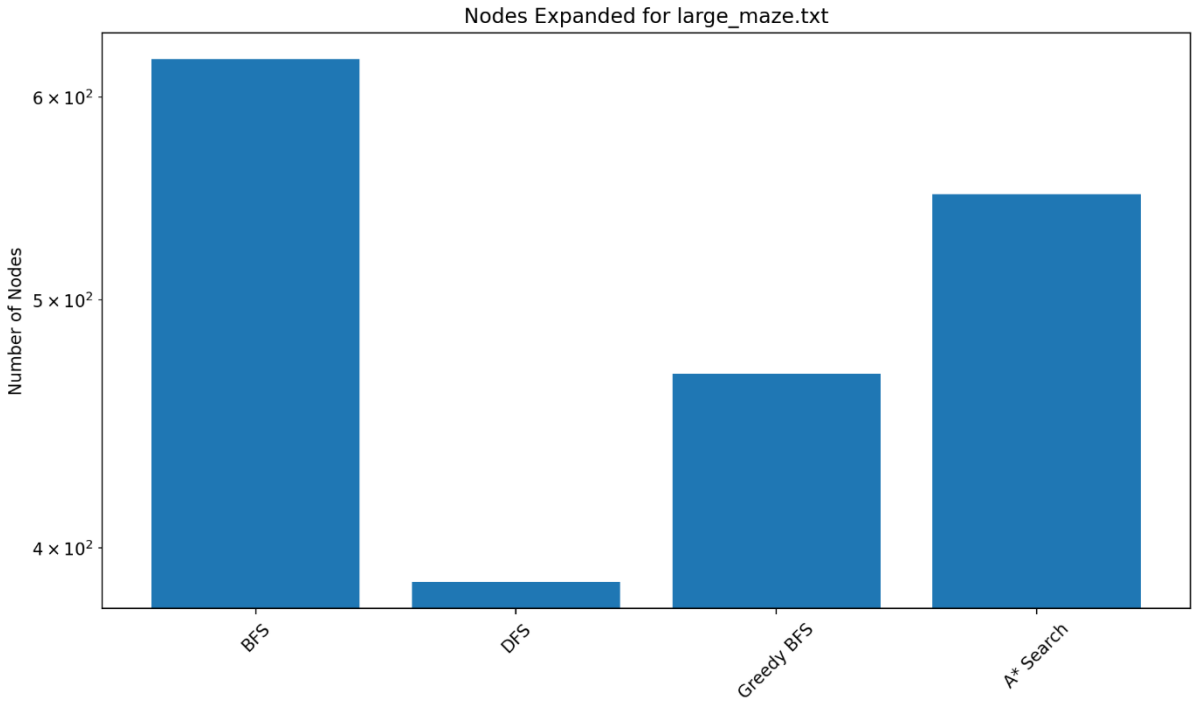

Nodes Expanded for small_maze.txt
Number of Nodes
6×10^1
4×10^1
BFS
DFS
Greedy BFS
A* Search


Nodes Expanded for medium_maze.txt
Number of Nodes
2×10^2
10^2
BFS
DFS
Greedy BFS
A* Search

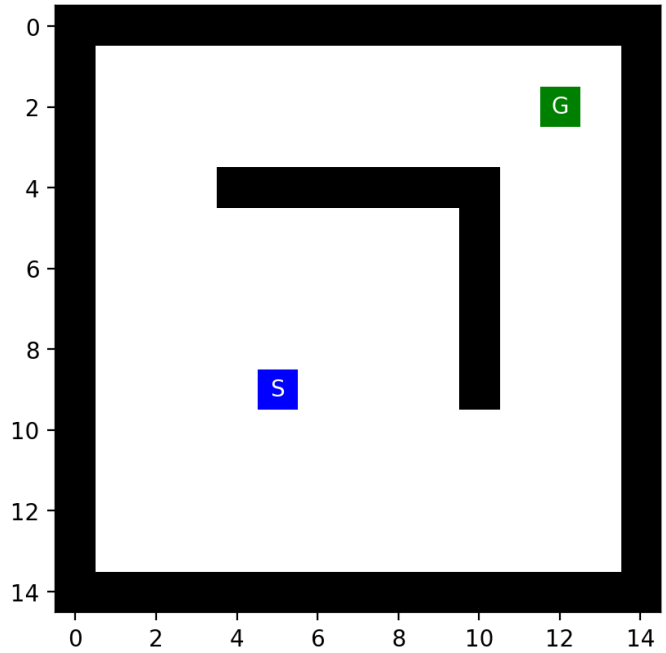


--- Results for large_maze.txt ---

	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	210	621	210	8	625	0.0085s
1	DFS	210	388	222	39	249	0.0135s
2	Greedy BFS	210	468	210	20	506	0.0068s
3	A* Search	210	550	210	12	564	0.0085s

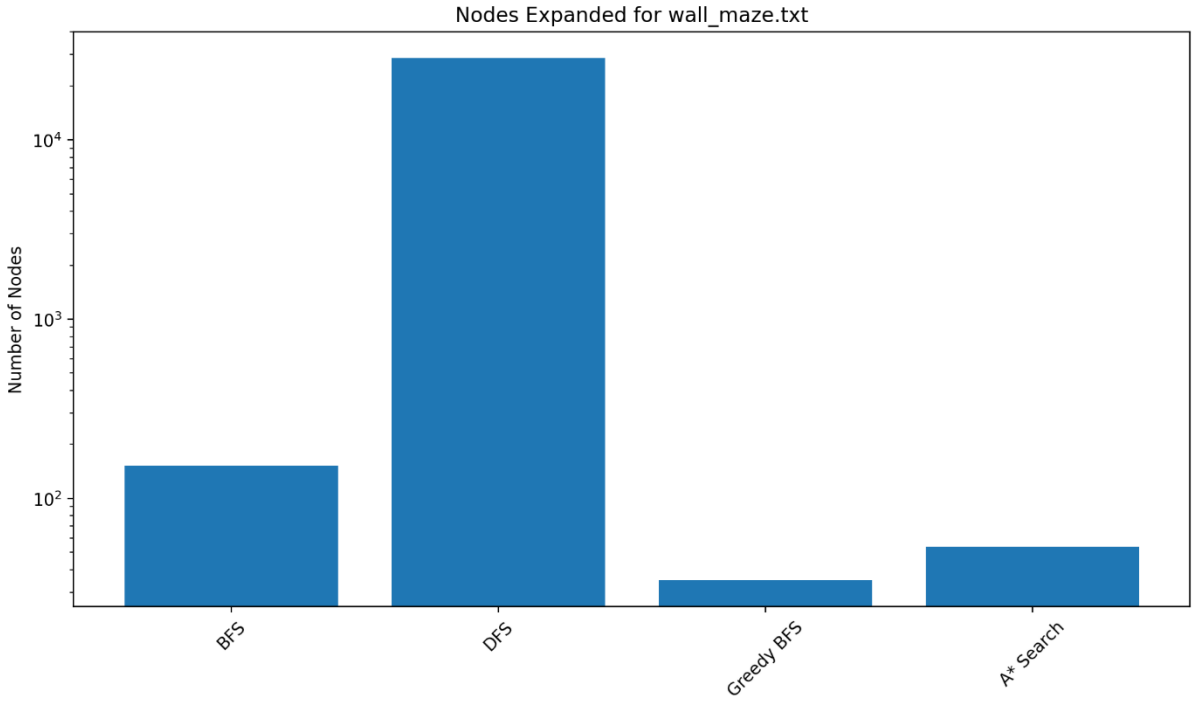


- L_maze.txt (hay còn gọi là **wall_maze**): Mê cung có một bức tường dài hình chữ L chắn giữa điểm bắt đầu và kết thúc. Đây là một "cái bẫy" kinh điển cho các thuật toán quá "tham lam" (như GBFS), vì chúng có thể bị kẹt lại khi cố gắng đi thẳng đến đích mà không nhận ra cần phải đi vòng.

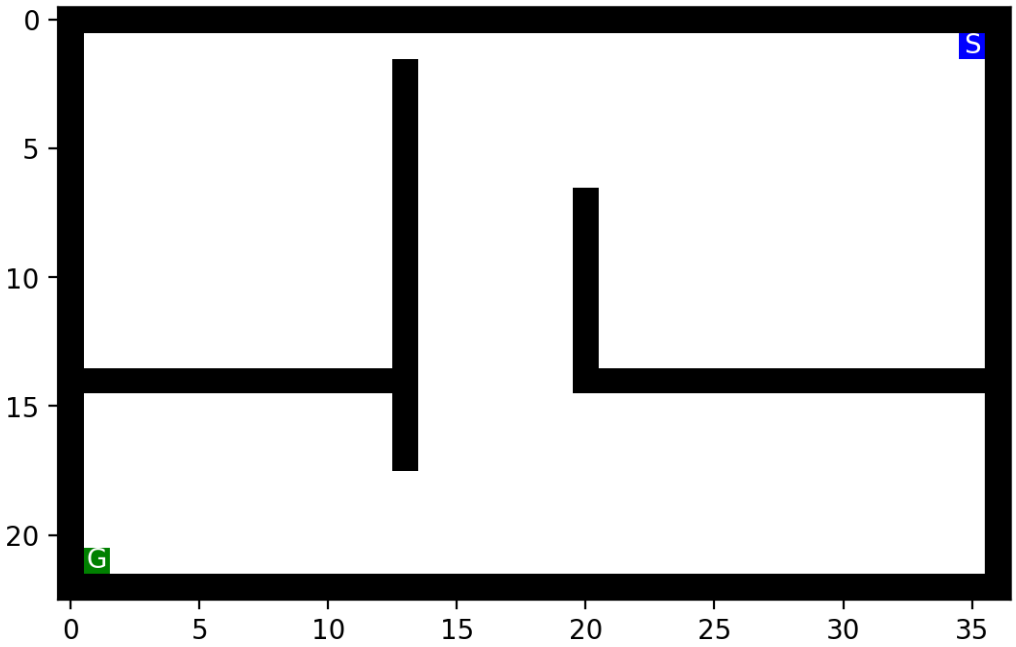


--- Results for wall_maze.txt ---

	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	16	152	16	18	158	0.0018s
1	DFS	82	28684	98	97	168	0.6255s
2	Greedy BFS	24	35	24	25	83	0.0007s
3	A* Search	16	54	16	26	104	0.0013s

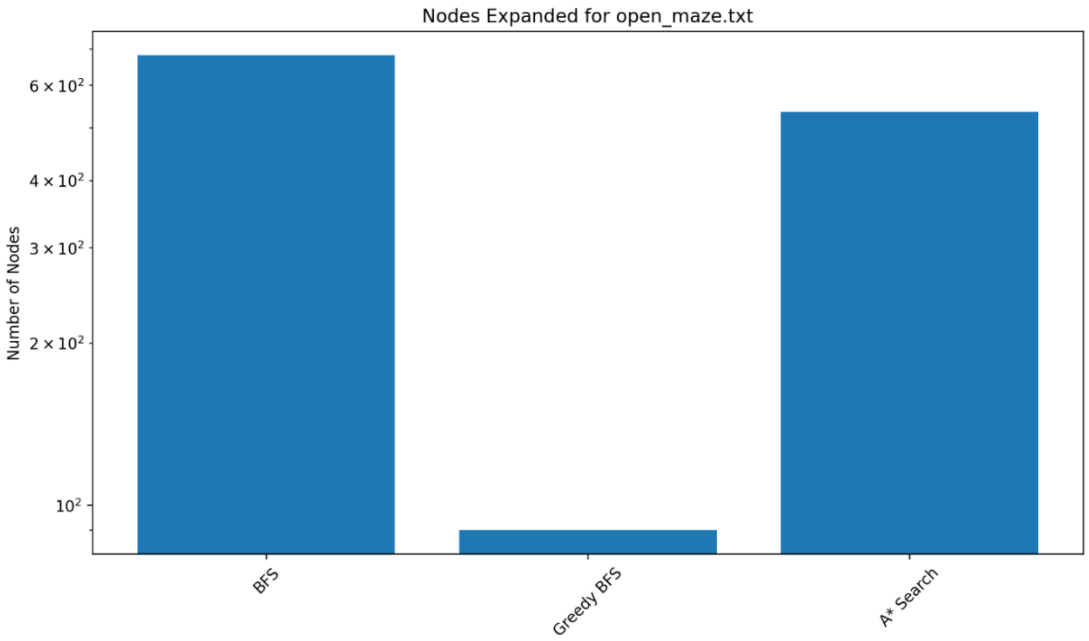


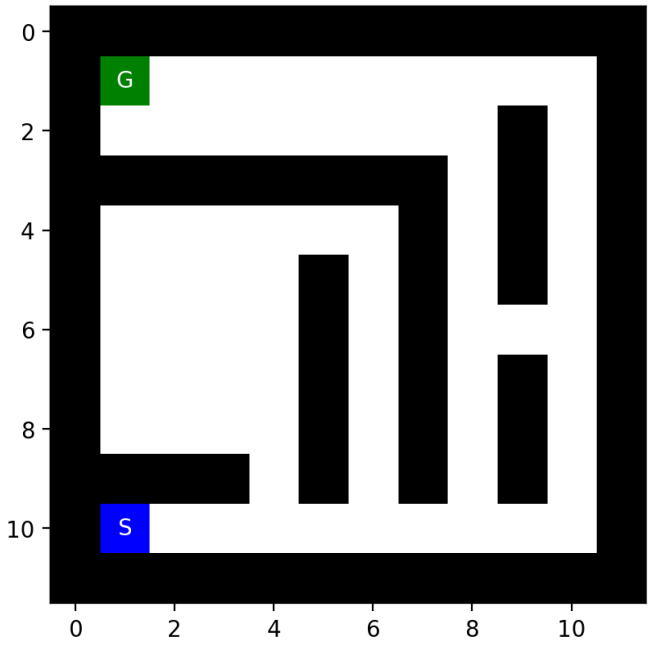
- Open_maze.txt và loops_maze.txt: Mê cung có các vòng lặp. Thử thách này đặc biệt quan trọng để kiểm tra cơ chế xử lý các trạng thái đã duyệt (sử dụng tập reached). Nếu không có cơ chế này, các thuật toán như DFS có thể rơi vào vòng lặp vô hạn.



--- Results for open_maze.txt ---

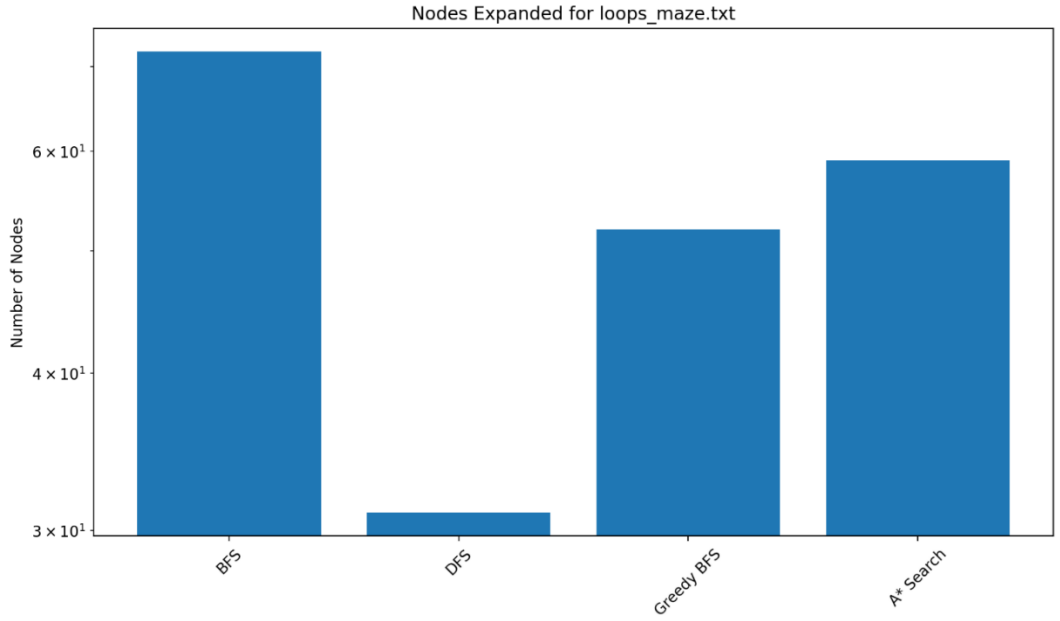
	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	54	683	54	25	685	0.0136s
1	DFS	N/A	N/A	N/A	N/A	N/A	30.0005s
2	Greedy BFS	68	90	68	66	220	0.0016s
3	A* Search	54	536	54	28	574	0.0105s





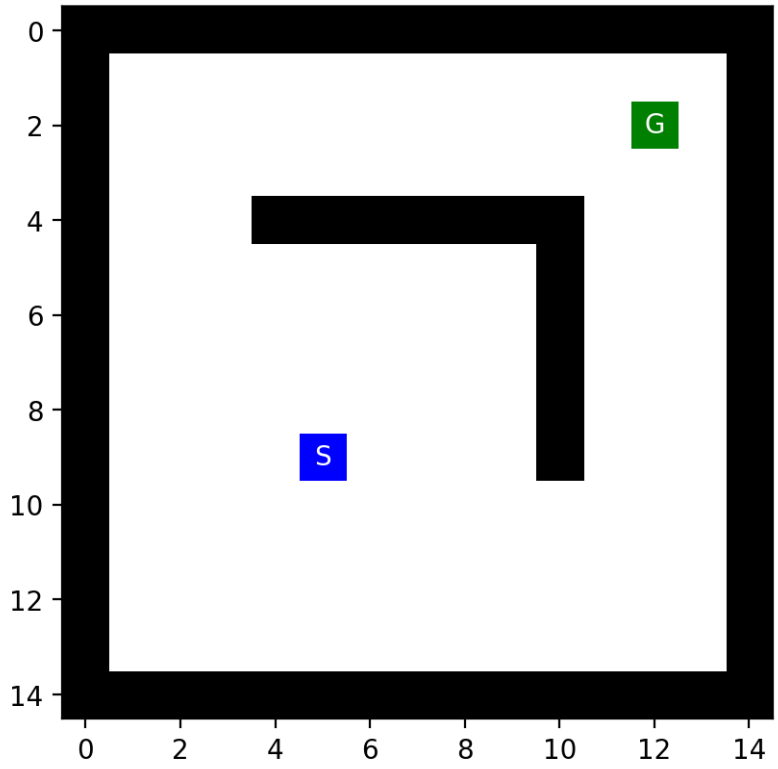
--- Results for loops_maze.txt ---

	Algorithm	Path Cost	Nodes Expanded	Max Tree Depth	Max Frontier Size	Max Nodes in Memory	Time
0	BFS	23	72	23	8	72	0.0017s
1	DFS	27	31	27	12	39	0.0006s
2	Greedy BFS	23	52	23	10	70	0.0009s
3	A* Search	23	59	23	6	65	0.0009s



2.3. So sánh hiệu năng: BFS và A* (Maze_BFS_vs_A_Star.ipynb)

File notebook này là trung tâm của bài thực hành, thực hiện một so sánh trực tiếp và định lượng giữa một thuật toán tìm kiếm mù (BFS) và một thuật toán tìm kiếm có thông tin (A*) trên cùng một mê cung thử thách là L_maze.txt.



Kết quả chi tiết

- BFS (Tìm kiếm theo chiều rộng):
    - Độ dài đường đi: 16
    - Số ô đã duyệt (states explored): 151
    - Thời gian thực thi: Khoảng 30.2 ms
    - Nhận xét: BFS hoạt động giống như việc đổ nước vào mê cung từ điểm bắt đầu, loang ra mọi ngóc ngách theo từng lớp. Do đó, nó đảm bảo tìm ra đường đi ngắn nhất (tối ưu về số bước). Tuy nhiên, phương pháp này rất không hiệu quả vì nó không có "ý thức" về hướng của mục tiêu, dẫn đến việc lãng phí tài nguyên để khám phá những khu vực hoàn toàn không liên quan đến lời giải.
- A* Search (Sử dụng Heuristic là Khoảng cách Manhattan):

- Độ dài đường đi: 16
- Số ô đã duyệt (states explored): 66
- Thời gian thực thi: Khoảng 6.93 ms
- Nhận xét: A* cũng tìm ra đường đi ngắn nhất, chứng tỏ tính tối ưu của nó. Sự vượt trội nằm ở hiệu quả: nhờ có heuristic "dẫn đường", A* ưu tiên khám phá các ô không chỉ gần điểm xuất phát mà còn gần cả điểm đích. Quá trình tìm kiếm của nó trông giống như một "chùm tia" tập trung hướng về mục tiêu, bỏ qua các nhánh rẽ vô ích. Điều này giúp giảm đáng kể số lượng trạng thái cần duyệt (chỉ bằng khoảng 29% so với BFS) và tăng tốc độ tìm kiếm.

Phân tích sâu hơn

Kết quả thực nghiệm này là một minh chứng rõ ràng cho sức mạnh của việc sử dụng thông tin trong tìm kiếm. Sự hiệu quả của A* đến từ cách nó cân bằng một cách thông minh giữa hai yếu tố:

1. Chi phí đã đi (g(n)): Yếu tố này đảm bảo A* không đi quá xa vào một con đường dài và tốn kém. Nó có vai trò tương tự như trong BFS/UCS, giữ cho việc tìm kiếm có tính tối ưu.
2. Chi phí ước tính (h(n)): Yếu tố này cung cấp "tầm nhìn xa", giúp thuật toán ưu tiên các hướng đi có vẻ hứa hẹn nhất. Đây là thành phần "thông minh" của A*.

Sự kết hợp này giúp A* tránh được sự "mù quáng" của BFS và sự "thiển cận" của GBFS (vốn chỉ quan tâm đến h(n)), tạo nên một trong những thuật toán tìm đường toàn diện và hiệu quả nhất.

2.4. Khảo sát về Heuristics (Explore_heuristics.ipynb)

Heuristic là "trái tim và linh hồn" của các thuật toán tìm kiếm có thông tin. Chất lượng của heuristic ảnh hưởng trực tiếp đến hiệu năng của thuật toán. Một hàm heuristic tốt h(n) phải thỏa mãn điều kiện **admissible** (chấp nhận được), nghĩa là nó không bao giờ đánh giá quá cao chi phí thực tế để đi từ trạng thái n đến mục tiêu. Điều này là tiên quyết để đảm bảo A* tìm ra lời giải tối ưu.

File Explore_heuristics.ipynb (viết bằng ngôn ngữ R) đã thực hiện một cuộc khảo sát trực quan về 3 loại heuristic khoảng cách phổ biến:

1. **Khoảng cách Manhattan (Manhattan distance):** h(n) = |x1 - x2| + |y1 - y2|. Được đặt tên theo cách di chuyển trên các con đường vuông góc của quận Manhattan, New York. Đây là heuristic hoàn hảo cho bài toán mê cung vì tác nhân chỉ có thể di chuyển theo chiều ngang và dọc. Trong một mê cung không có vật cản, giá trị của nó chính xác bằng chi phí thực tế.
2. **Khoảng cách Euclid (Euclidean distance):** h(n) = sqrt((x1 - x2)^2 + (y1 - y2)^2). Đây là khoảng cách đường chim bay. Vì tác nhân không thể đi chéo, khoảng cách này luôn nhỏ hơn hoặc bằng khoảng cách Manhattan, do đó nó cũng là một heuristic admissible. Tuy nhiên, nó đánh giá thấp chi phí thực tế hơn, làm cho A* phải khám phá nhiều node hơn một chút so với khi dùng Manhattan.

Notebook này trực quan hóa các giá trị heuristic dưới dạng bản đồ nhiệt, cho thấy rằng đối với bài toán di chuyển trên lưới 4 hướng, khoảng cách Manhattan phản ánh đúng nhất "chi phí" di chuyển. Nó cung cấp một ước tính chặt chẽ nhất mà không vi phạm tính admissible, do đó nó là heuristic hiệu quả nhất cho thuật toán A* trong bối cảnh này.

3. Kết luận

Qua bài thực hành, đã triển khai, đánh giá và so sánh thành công các thuật toán tìm kiếm cơ bản để giải quyết bài toán tìm đường trong mê cung. Các kết quả thực nghiệm đã tái khẳng định và làm sáng tỏ các điểm lý thuyết quan trọng trong lĩnh vực Trí tuệ Nhân tạo:

- **Sự đánh đổi của Tìm kiếm mù:** Các thuật toán như BFS có ưu điểm là đơn giản, dễ triển khai, và đảm bảo tính tối ưu (về số bước), nhưng phải trả giá bằng hiệu suất tính toán kém. Chúng không có khả năng mở rộng tốt khi không gian trạng thái trở nên lớn hơn vì chúng khám phá một cách "mù quáng".
- **Sức mạnh của Heuristics:** Các thuật toán tìm kiếm có thông tin, đặc biệt là A*, đã chứng tỏ sự vượt trội rõ rệt. Bằng cách sử dụng một hàm heuristic hợp lý để dẫn đường, chúng có thể tập trung nỗ lực tính toán vào những vùng hứa hẹn nhất của không gian tìm kiếm, giúp giảm đáng kể số lượng trạng thái cần duyệt và thời gian thực thi.
- **Tầm quan trọng của việc thiết kế Heuristic:** Hiệu năng của A* phụ thuộc trực tiếp vào chất lượng của hàm heuristic. Việc lựa chọn một hàm heuristic tốt (vừa admissible, vừa cung cấp ước tính càng gần với chi phí thực tế càng tốt) là yếu tố then chốt. Khoảng cách Manhattan là một lựa chọn xuất sắc cho các bài toán di chuyển trên lưới.

Nhìn chung, bài thực hành đã cung cấp một cái nhìn sâu sắc và thực tế về cách áp dụng các thuật toán tìm kiếm để giải quyết vấn đề. Những khái niệm này không chỉ giới hạn trong việc giải mê cung mà còn là nền tảng cho vô số ứng dụng thực tế khác như hệ thống định vị GPS, lập kế hoạch cho robot, giải các câu đố logic, và tối ưu hóa chuỗi cung ứng.

Phụ lục:

```
class Node: # Bổ sung phương thức so sánh (__lt__ cho "less than") vào lớp Node
        def __lt__(self, other):
            """Phương thức so sánh cần thiết cho heapq."""
            return self.cost < other.cost
# Your code goes here
from collections import deque
import maze_helper as mh
import numpy as np
import heapq
import pandas as pd
import time


def breadth_first_search(maze, timeout=None):
    """Sử dụng thuật toán tìm kiếm theo chiều rộng (BFS)."""
    start_pos = mh.find_pos(maze, 'S')
    goal_pos = mh.find_pos(maze, 'G')
    start_time = time.time()


    # Node ban đầu
    start_node = Node(pos=start_pos, parent=None, action=None, cost=0)
```

```
# Frontier sử dụng hàng đợi (deque)
frontier = deque([start_node])
reached = {start_node.pos}

# Biến thống kê
nodes_expanded = 0
max_tree_depth = 0
max_frontier_size = 1

# Bắt đầu tìm kiếm
while frontier:
    if timeout and (time.time() - start_time) > timeout:
        return None
    max_frontier_size = max(max_frontier_size, len(frontier))

    # Lấy node từ đầu hàng đợi
    node = frontier.popleft()
    nodes_expanded += 1

    # Cập nhật độ sâu tối đa
    max_tree_depth = max(max_tree_depth, node.cost)

    # Kiểm tra xem có phải là đích không
    if node.pos == goal_pos:
        return {
            "solution_node": node,
            "nodes_expanded": nodes_expanded,
```

```
            "max_tree_depth": max_tree_depth,
            "max_frontier_size": max_frontier_size,
            "max_nodes_in_memory": len(reached) + len(frontier)
        }

    # Mở rộng các node con
    for action, pos in get_neighbors(maze, node.pos):
        if pos not in reached:
            child = Node(pos=pos, parent=node, action=action, cost=node.cost + 1)
            frontier.append(child)
            reached.add(pos)

  return None # Không tìm thấy lời giải

def get_neighbors(maze, pos):
  """Hàm trợ giúp để lấy các ô hàng xóm hợp lệ."""
  row, col = pos
  neighbors = []
  # Các hành động có thể: Lên, Xuống, Trái, Phải
  actions = [('Up', (-1, 0)), ('Down', (1, 0)), ('Left', (0, -1)), ('Right', (0, 1))]
  for action, (dr, dc) in actions:
      new_row, new_col = row + dr, col + dc
      if 0 <= new_row < maze.shape[0] and 0 <= new_col < maze.shape[1] and maze[new_row,
new_col] != 'X':
          neighbors.append((action, (new_row, new_col)))
  return neighbors
```

```
def depth_first_search(maze, timeout=None):
    """Sử dụng thuật toán tìm kiếm theo chiều sâu (DFS) với kiểm tra chu trình."""
    start_pos = mh.find_pos(maze, 'S')
    goal_pos = mh.find_pos(maze, 'G')
    start_time = time.time()

    start_node = Node(pos=start_pos, parent=None, action=None, cost=0)

    # Frontier sử dụng ngăn xếp (stack)
    frontier = [start_node]

    # Biến thống kê
    nodes_expanded = 0
    max_tree_depth = 0
    max_frontier_size = 1

    while frontier:
        if timeout and (time.time() - start_time) > timeout:
            return None
        max_frontier_size = max(max_frontier_size, len(frontier))

        # Lấy node từ đỉnh ngăn xếp
        node = frontier.pop()
        nodes_expanded += 1

        max_tree_depth = max(max_tree_depth, node.cost)
```

```
        if node.pos == goal_pos:
            path = node.get_path_from_root()
            return {
                "solution_node": node,
                "nodes_expanded": nodes_expanded,
                "max_tree_depth": max_tree_depth,
                "max_frontier_size": max_frontier_size,
                "max_nodes_in_memory": len(path) + len(frontier) # Chỉ lưu trữ đường đi hiện tại và frontier
            }

        # Cycle checking: Kiểm tra xem node con đã tồn tại trên đường đi chưa
        path_to_current = {n.pos for n in node.get_path_from_root()}

        for action, pos in get_neighbors(maze, node.pos):
            if pos not in path_to_current:
                child = Node(pos=pos, parent=node, action=action, cost=node.cost + 1)
                frontier.append(child)

    return None
# Your code goes here
def manhattan_distance(pos1, pos2):
    """Tính khoảng cách Manhattan giữa hai điểm."""
    return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1])

def a_star_search(maze, heuristic_func, timeout=None):
    """Sử dụng thuật toán A*."""
```

```
start_pos = mh.find_pos(maze, 'S')
goal_pos = mh.find_pos(maze, 'G')
start_time = time.time()

start_node = Node(pos=start_pos, parent=None, action=None, cost=0)

# Frontier là một hàng đợi ưu tiên (min-heap)
# (priority, node)
frontier = [(0, start_node)]
reached = {start_node.pos: 0} # Lưu trữ chi phí tốt nhất đến mỗi vị trí

nodes_expanded = 0
max_tree_depth = 0
max_frontier_size = 1

while frontier:
    if timeout and (time.time() - start_time) > timeout:
        return None
    max_frontier_size = max(max_frontier_size, len(frontier))

    # Lấy node có f(n) thấp nhất
    priority, node = heapq.heappop(frontier)

    # Nếu đã có đường đi tốt hơn đến vị trí này, bỏ qua
    if node.cost > reached[node.pos]:
        continue
```

```
        nodes_expanded += 1
        max_tree_depth = max(max_tree_depth, node.cost)

        if node.pos == goal_pos:
            return {
                "solution_node": node,
                "nodes_expanded": nodes_expanded,
                "max_tree_depth": max_tree_depth,
                "max_frontier_size": max_frontier_size,
                "max_nodes_in_memory": len(reached) + len(frontier)
            }

        for action, pos in get_neighbors(maze, node.pos):
            new_cost = node.cost + 1
            if pos not in reached or new_cost < reached[pos]:
                reached[pos] = new_cost
                priority = new_cost + heuristic_func(pos, goal_pos)
                child = Node(pos=pos, parent=node, action=action, cost=new_cost)
                heapq.heappush(frontier, (priority, child))

    return None

def greedy_best_first_search(maze, heuristic_func, timeout=None):
    """Sử dụng thuật toán Greedy Best-First Search."""
    start_pos = mh.find_pos(maze, 'S')
    goal_pos = mh.find_pos(maze, 'G')
    start_time = time.time()
```

```
start_node = Node(pos=start_pos, parent=None, action=None, cost=0)

# Frontier là hàng đợi ưu tiên, sắp xếp theo heuristic
frontier = [(heuristic_func(start_pos, goal_pos), start_node)]
reached = {start_node.pos}

nodes_expanded = 0
max_tree_depth = 0
max_frontier_size = 1

while frontier:
    if timeout and (time.time() - start_time) > timeout:
        return None
    max_frontier_size = max(max_frontier_size, len(frontier))

    priority, node = heapq.heappop(frontier)

    nodes_expanded += 1
    max_tree_depth = max(max_tree_depth, node.cost)

    if node.pos == goal_pos:
        return {
            "solution_node": node,
            "nodes_expanded": nodes_expanded,
            "max_tree_depth": max_tree_depth,
            "max_frontier_size": max_frontier_size,
```

```
            "max_nodes_in_memory": len(reached) + len(frontier)
        }

    for action, pos in get_neighbors(maze, node.pos):
        if pos not in reached:
            reached.add(pos)
            priority = heuristic_func(pos, goal_pos)
            child = Node(pos=pos, parent=node, action=action, cost=node.cost + 1)
            heapq.heappush(frontier, (priority, child))

  return None
# Add code

def solve_and_report(maze_files, algorithms, timeout=30):
  results = {}
  for file in maze_files:
    print(f"--- Solving: {file} ---")
    with open(file, "r") as f:
        maze = mh.parse_maze(f.read())

    maze_results = []
    for name, func in algorithms.items():
        start_time = time.time()
        if "A*" in name or "Greedy" in name:
            result = func(maze, manhattan_distance, timeout=timeout)
        else:
            result = func(maze, timeout=timeout)
```

```
        duration = time.time() - start_time

        if result:
            solution_node = result['solution_node']
            path_cost = solution_node.cost
            nodes_expanded = result['nodes_expanded']
            max_depth = result['max_tree_depth']
            max_frontier = result['max_frontier_size']
            max_memory = result['max_nodes_in_memory']

            maze_results.append([name, path_cost, nodes_expanded, max_depth, max_frontier,
max_memory, f"{duration:.4f}s"])

            # Visualize
            path_nodes = solution_node.get_path_from_root()
            maze_viz = np.copy(maze)
            for node in path_nodes:
                if maze_viz[node.pos] == ' ':
                    maze_viz[node.pos] = 'P'
            print(f"\nSolution for {name}:")
            mh.show_maze(maze_viz)

        else:
            maze_results.append([name, "N/A", "N/A", "N/A", "N/A", "N/A",
f"{duration:.4f}s"])
            print(f"\nSolution for {name}: Không tìm thấy lời giải (timeout).")
```

```
        df = pd.DataFrame(maze_results, columns=["Algorithm", "Path Cost", "Nodes Expanded", "Max Tree Depth", "Max Frontier Size", "Max Nodes in Memory", "Time"])
        results[file] = df

    return results

maze_files = ["small_maze.txt", "medium_maze.txt", "large_maze.txt",
              "open_maze.txt", "wall_maze.txt", "loops_maze.txt",
              "empty_maze.txt", "empty_maze_2.txt"]

algorithms = {
    "BFS": breadth_first_search,
    "DFS": depth_first_search,
    "Greedy BFS": greedy_best_first_search,
    "A* Search": a_star_search
}

all_results = solve_and_report(maze_files, algorithms)

for maze_name, df in all_results.items():
    print(f"\n--- Results for {maze_name} ---")
    display(df)

%time
# Add charts
import matplotlib.pyplot as plt
```

```
for maze_name, df in all_results.items():
    df_numeric = df[pd.to_numeric(df['Nodes Expanded'], errors='coerce').notna()]
    if df_numeric.empty:
        continue

    plt.figure(figsize=(10, 6))
    plt.bar(df_numeric['Algorithm'], pd.to_numeric(df_numeric['Nodes Expanded']))
    plt.title(f'Nodes Expanded for {maze_name}')
    plt.ylabel('Number of Nodes')
    plt.yscale('log') # Thang đo log có thể hữu ích cho các giá trị chênh lệch lớn
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

Ngoài ra còn thêm iterative deepening search

IDS

```
# Your code/answer goes here
def depth_limited_search(maze, limit, timeout=None):
    start_pos = mh.find_pos(maze, 'S')
    goal_pos = mh.find_pos(maze, 'G')
    start_time = time.time()

    start_node = Node(pos=start_pos, parent=None, action=None, cost=0)
    frontier = [start_node]

    nodes_expanded = 0
    max_frontier_size = 1
```

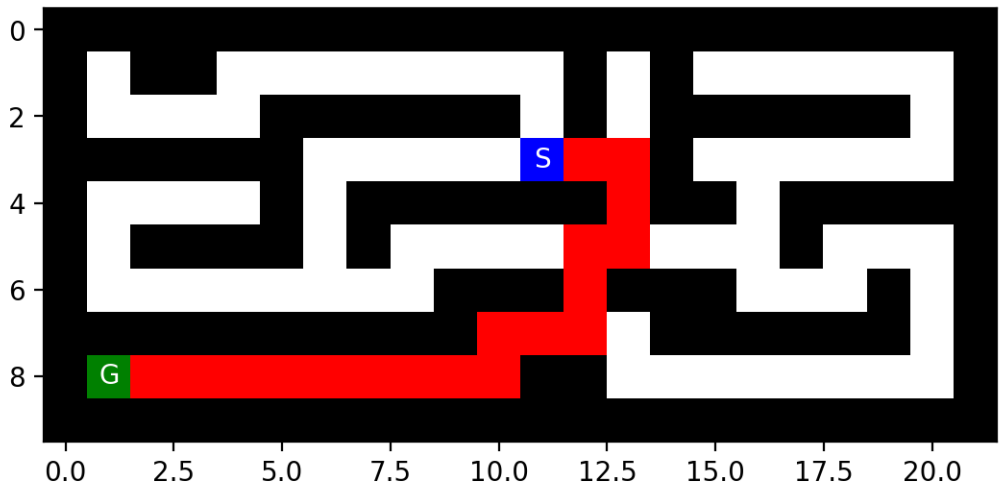
```
while frontier:
    if timeout and (time.time() - start_time) > timeout:
        return None
    max_frontier_size = max(max_frontier_size, len(frontier))
    node = frontier.pop()
    nodes_expanded += 1

    if node.pos == goal_pos:
        return {
            "solution_node": node,
            "nodes_expanded": nodes_expanded,
            "max_frontier_size": max_frontier_size,
            "cutoff": False
        }

    if node.cost >= limit:
        continue # Dừng nhánh này

    path_to_current = {n.pos for n in node.get_path_from_root()}

    for action, pos in get_neighbors(maze, node.pos):
        if pos not in path_to_current:
            child = Node(pos=pos, parent=node, action=action, cost=node.cost + 1)
            frontier.append(child)
```

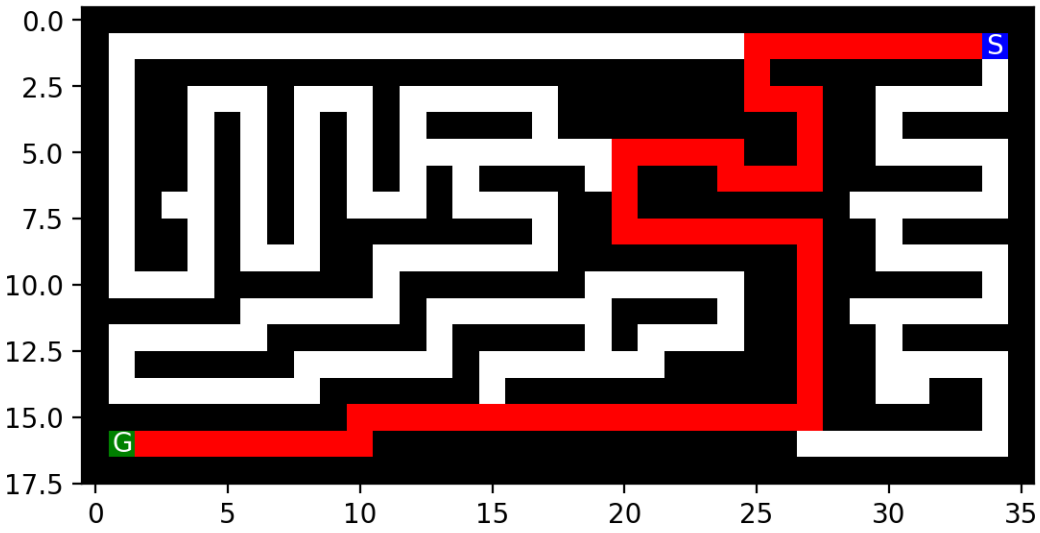
```
    return {"cutoff": True, "nodes_expanded": nodes_expanded, "max_frontier_size":
max_frontier_size}


def iterative_deepening_search(maze, timeout=None):
    """Sử dụng thuật toán tìm kiếm sâu dần (IDS)."""
    total_nodes_expanded = 0
    total_max_frontier = 0
    start_time = time.time()

    for depth in range(maze.size): # Giới hạn độ sâu tối đa là tổng số ô
        if timeout and (time.time() - start_time) > timeout:
            return None

        result = depth_limited_search(maze, depth, timeout=timeout)

        if result is None:
            return None

        total_nodes_expanded += result.get('nodes_expanded', 0)
        total_max_frontier = max(total_max_frontier, result.get('max_frontier_size', 0))

        if not result.get("cutoff"):
            if result.get("solution_node"):
                solution_node = result['solution_node']
                return {
                    "solution_node": solution_node,
                    "nodes_expanded": total_nodes_expanded,
```

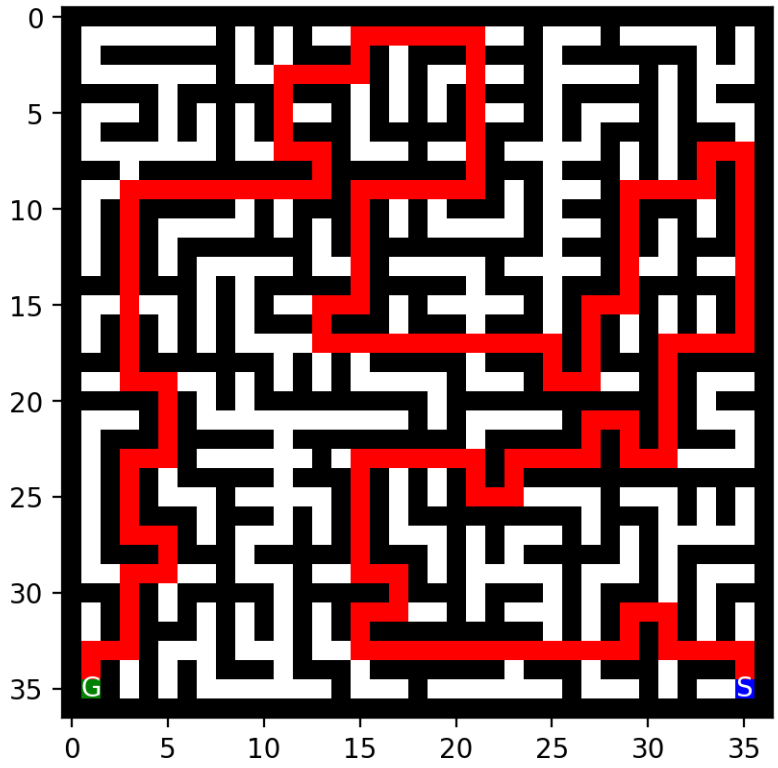
```
                "max_tree_depth": solution_node.cost,
                "max_frontier_size": total_max_frontier,
                # Không gian bộ nhớ của IDS tương tự DFS, tỉ lệ với độ sâu
                "max_nodes_in_memory": (solution_node.cost + 1) * 4
            }
    return None

# Add code

def solve_and_report(maze_files, algorithms, timeout=30):
    results = {}
    for file in maze_files:
        print(f"--- Solving: {file} ---")
        with open(file, "r") as f:
            maze = mh.parse_maze(f.read())

        maze_results = []
        for name, func in algorithms.items():
            start_time = time.time()
            if name == "IDS":
                result = func(maze, timeout=timeout)
            duration = time.time() - start_time

            if result:
                solution_node = result['solution_node']
                path_cost = solution_node.cost
                nodes_expanded = result['nodes_expanded']
```

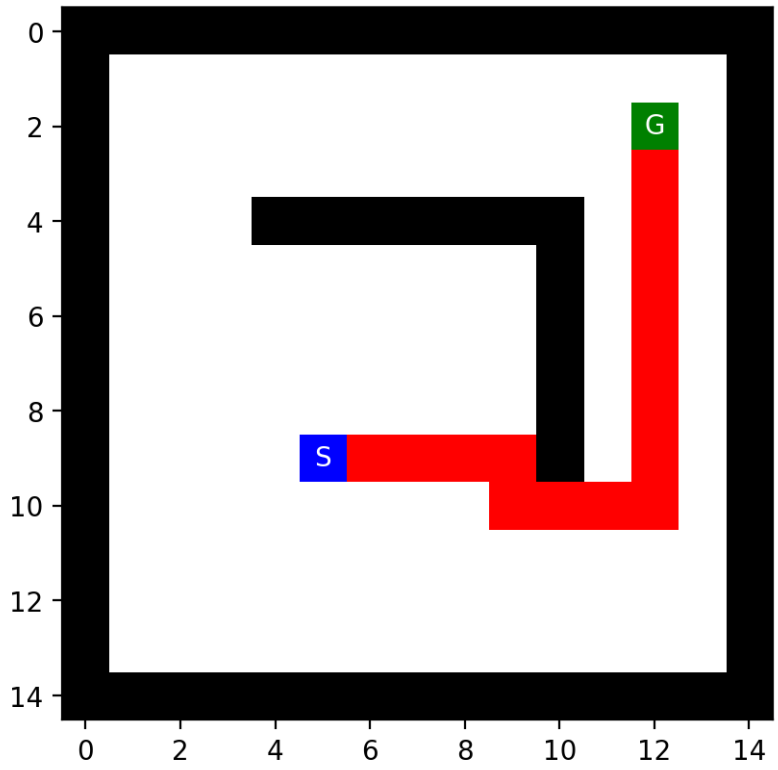
```
            max_depth = result['max_tree_depth']
            max_frontier = result['max_frontier_size']
            max_memory = result['max_nodes_in_memory']

            maze_results.append([name, path_cost, nodes_expanded, max_depth, max_frontier,
max_memory, f"{duration:.4f}s"])

            # Visualize
            path_nodes = solution_node.get_path_from_root()
            maze_viz = np.copy(maze)
            for node in path_nodes:
                if maze_viz[node.pos] == ' ':
                    maze_viz[node.pos] = 'P'
            print(f"\nSolution for {name}:")
            mh.show_maze(maze_viz)

        else:
            maze_results.append([name, "N/A", "N/A", "N/A", "N/A", "N/A", f"{duration:.4f}s"])
            print(f"\nSolution for {name}: Không tìm thấy lời giải (timeout).")

    df = pd.DataFrame(maze_results, columns=["Algorithm", "Path Cost", "Nodes Expanded",
"Max Tree Depth", "Max Frontier Size", "Max Nodes in Memory", "Time"])
    results[file] = df

  return results

maze_files = ["small_maze.txt", "medium_maze.txt", "large_maze.txt",
        "open_maze.txt", "wall_maze.txt", "loops_maze.txt",
```

```
        "empty_maze.txt", "empty_maze_2.txt"]

algorithms = {
    "IDS": iterative_deepening_search
}

all_results = solve_and_report(maze_files, algorithms)

for maze_name, df in all_results.items():
    print(f"\n--- Results for {maze_name} ---")
    display(df)

%time
```



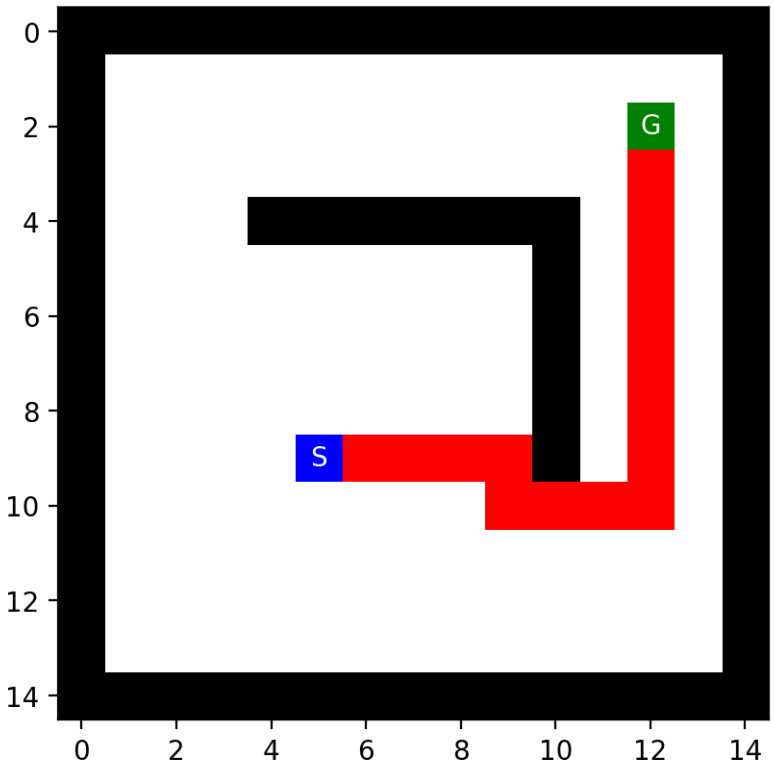

S
G
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
20
25
30
35

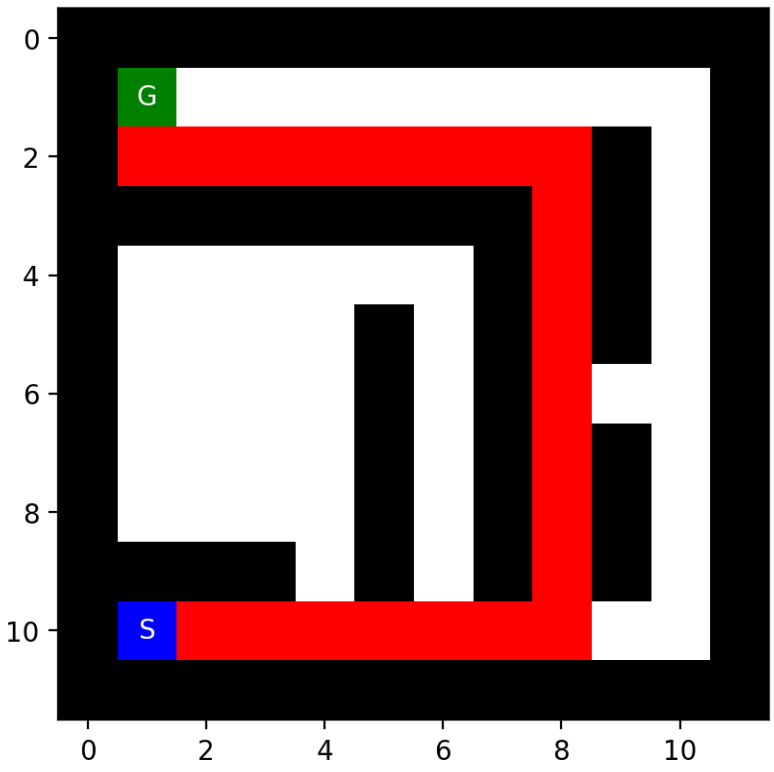

G
S
0
5
10
15
20
25
30
35

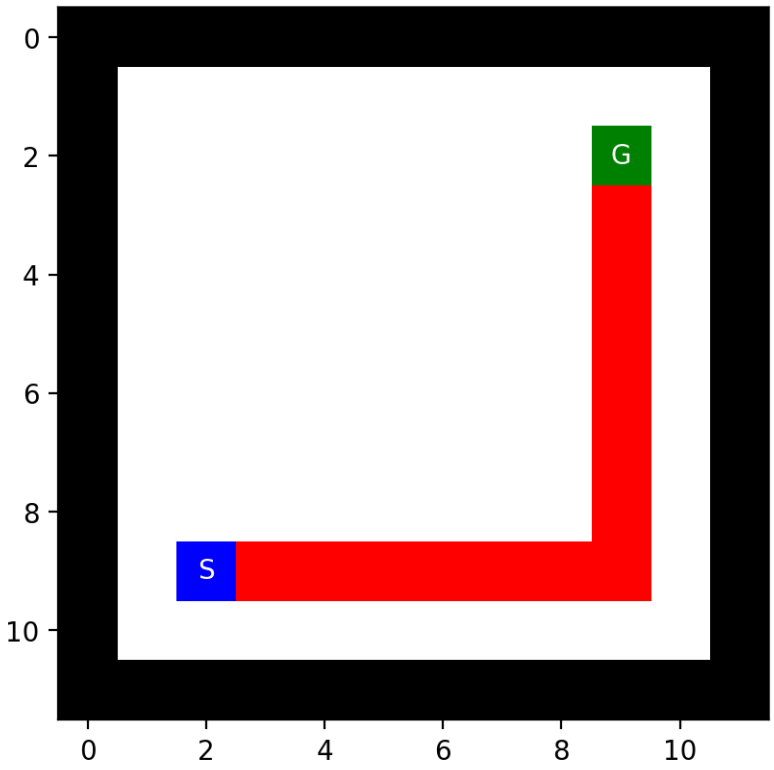


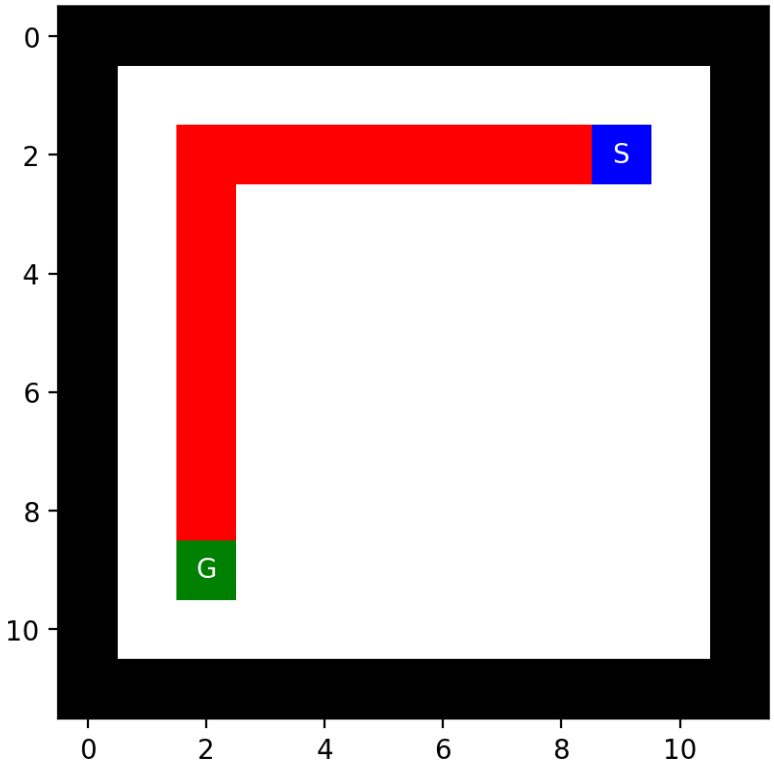
--- Solving: open_maze.txt --- **Solution for IDS: Không tìm thấy lời giải (timeout).**

```
--- Solving: open_maze.txt ---

Solution for IDS: Không tìm thấy lời giải (timeout).
```


G
S
0
2
4
6
8
10
12
14


G
S
0
2
4
6
8
10


G
S
0
2
4
6
8
10


S
G
0
2
4
6
8
10

```
--- Results for small_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory     Time
0       IDS         19            1057              19                  6                   80  0.0085s

--- Results for medium_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory     Time
0       IDS         68           16471              68                  8                  276  0.1941s

--- Results for large_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory     Time
0       IDS        210           60830             210                 39                  844  0.9527s

--- Results for open_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory      Time
0       IDS        N/A             N/A             N/A                N/A                  N/A  47.7648s

--- Results for wall_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory      Time
0       IDS         16         4426930              16                 28                   68  20.7350s

--- Results for loops_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory     Time
0       IDS         23           78925              23                 14                   96  0.5863s
```

```
--- Results for empty_maze.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory      Time
0       IDS         14          511704              14                 25                   60  1.9712s

--- Results for empty_maze_2.txt ---
  Algorithm  Path Cost  Nodes Expanded  Max Tree Depth  Max Frontier Size  Max Nodes in Memory     Time
0       IDS         14          601270              14                 26                   60  3.4744s
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.91 µs
```

Multiple Goals

 # Your code/answer goes here

```
def depth_limited_search(maze, limit):
    start_pos = mh.find_pos(maze, 'S')
    goal_pos = mh.find_pos(maze, 'G')
```

```
start_node = Node(pos=start_pos, parent=None, action=None, cost=0)
frontier = [start_node]

nodes_expanded = 0
max_frontier_size = 1

while frontier:
    max_frontier_size = max(max_frontier_size, len(frontier))
    node = frontier.pop()
    nodes_expanded += 1

    if node.pos in set_of_goal_positions:
        return {
            "solution_node": node,
            "nodes_expanded": nodes_expanded,
            "max_frontier_size": max_frontier_size,
            "cutoff": False
        }

    if node.cost >= limit:
        continue # Dừng nhánh này

    path_to_current = {n.pos for n in node.get_path_from_root()}

    for action, pos in get_neighbors(maze, node.pos):
        if pos not in path_to_current:
```

```
            child = Node(pos=pos, parent=node, action=action, cost=node.cost + 1)
            frontier.append(child)

    return {"cutoff": True, "nodes_expanded": nodes_expanded, "max_frontier_size":
max_frontier_size}

def iterative_deepening_search_update(maze):
    """Sử dụng thuật toán tìm kiếm sâu dần (IDS)."""
    total_nodes_expanded = 0
    total_max_frontier = 0

    for depth in range(maze.size): # Giới hạn độ sâu tối đa là tổng số ô
        result = depth_limited_search(maze, depth)

        total_nodes_expanded += result.get('nodes_expanded', 0)
        total_max_frontier = max(total_max_frontier, result.get('max_frontier_size', 0))

        if not result.get("cutoff"):
            if result.get("solution_node"):
                solution_node = result['solution_node']
                return {
                    "solution_node": solution_node,
                    "nodes_expanded": total_nodes_expanded,
                    "max_tree_depth": solution_node.cost,
                    "max_frontier_size": total_max_frontier,
                    # Không gian bộ nhớ của IDS tương tự DFS, tỉ lệ với độ sâu
                    "max_nodes_in_memory": (solution_node.cost + 1) * 4
```

}

return None