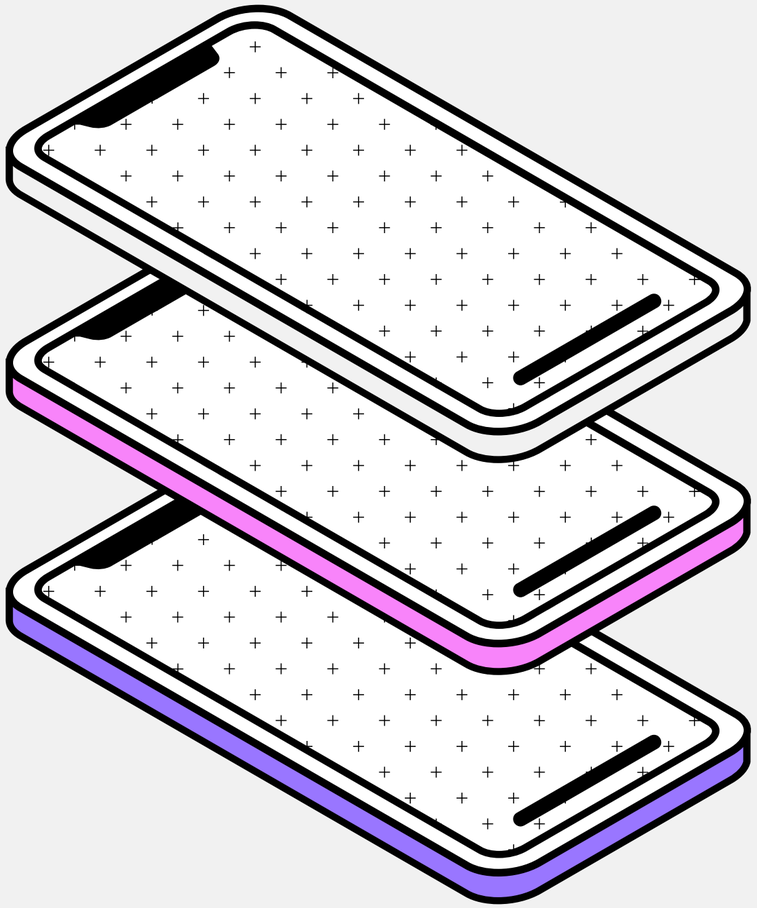
NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Sử dụng thuật toán A* để tìm đường đi ngắn nhất từ điểm xuất phát đến điểm đích trong một lưới không gian hai chiều có vật cản

NHÓM 6



Thành viên

Trương Chiến Nguyên - B22DCCN596

Phạm Lý Ngọc Đức - B22DCCN238

Hoàng Đình Phong - B22DCCN614

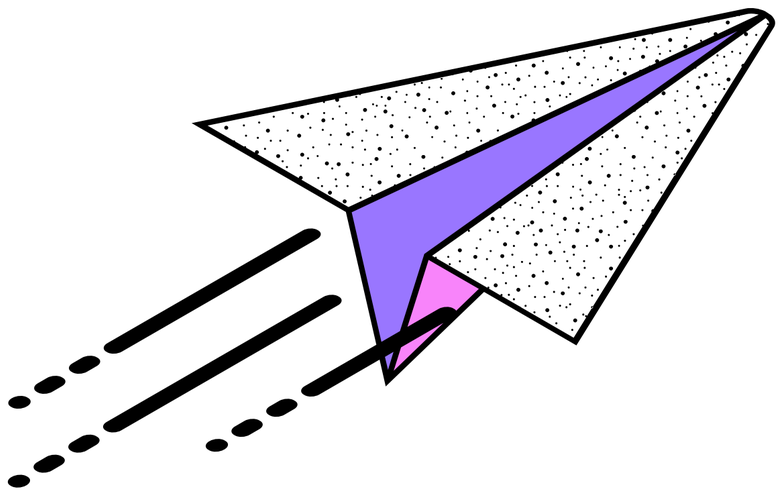
Hồ Lý Đức - B22DCCN226

Trần Thế Quang Minh - B22DCCN547

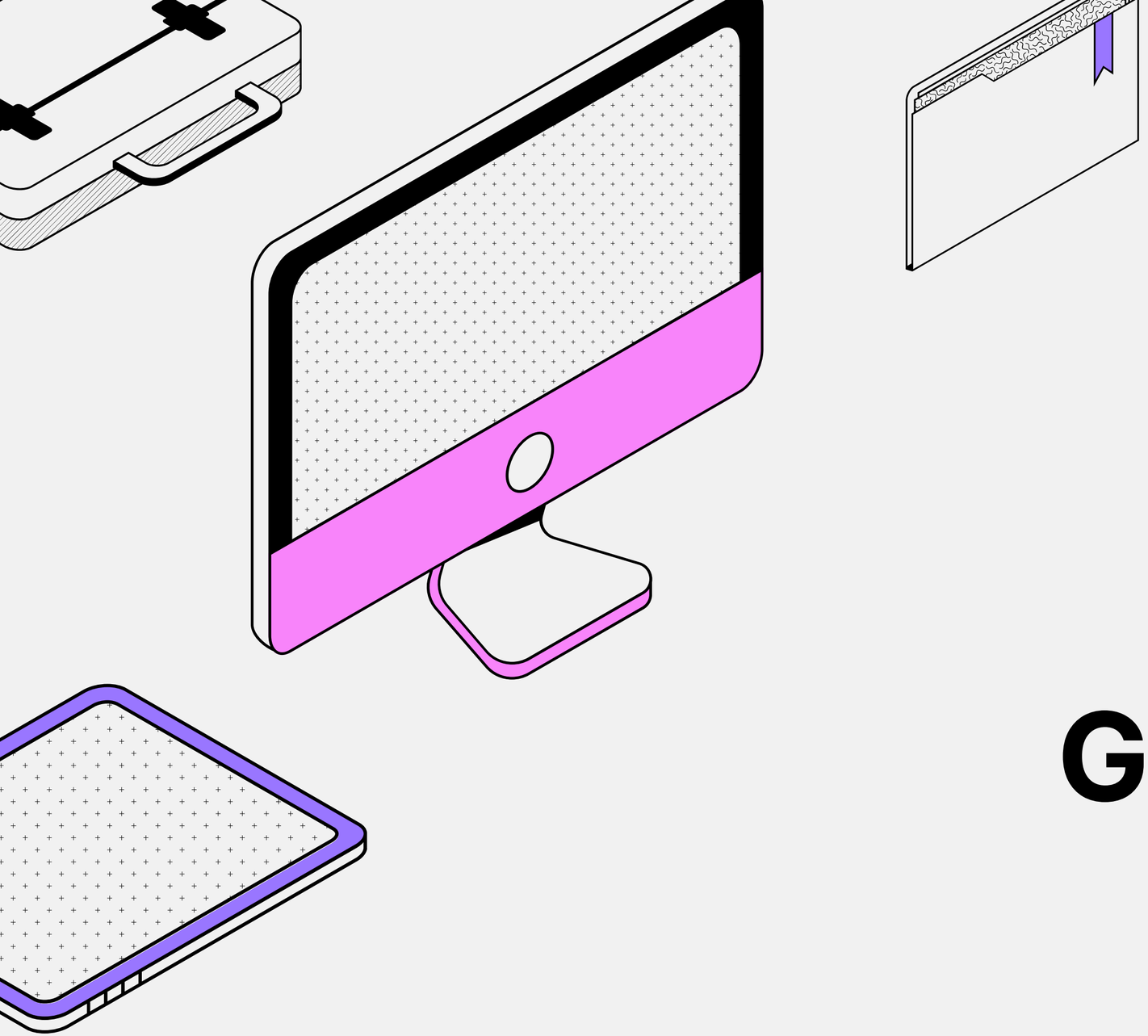
Nguyễn Ngọc Quang - B22DCCN646

Hà Duy Hiếu - B22DCCN306

A* Algorithm



- Giới thiệu
- Mô tả về thuật toán
- Đánh giá ưu và nhược điểm
- So sánh
- Ứng dụng
- Demo



Giới thiệu

Giới Thiệu Về A*

A* là thuật toán tìm kiếm trên đồ thị có trọng số, nhằm tìm đường đi ngắn nhất từ một node bắt đầu (start node) đến một node đích (goal node), sao cho tổng chi phí của đường đi là nhỏ nhất

Giới thiệu bài toán

Bài toán

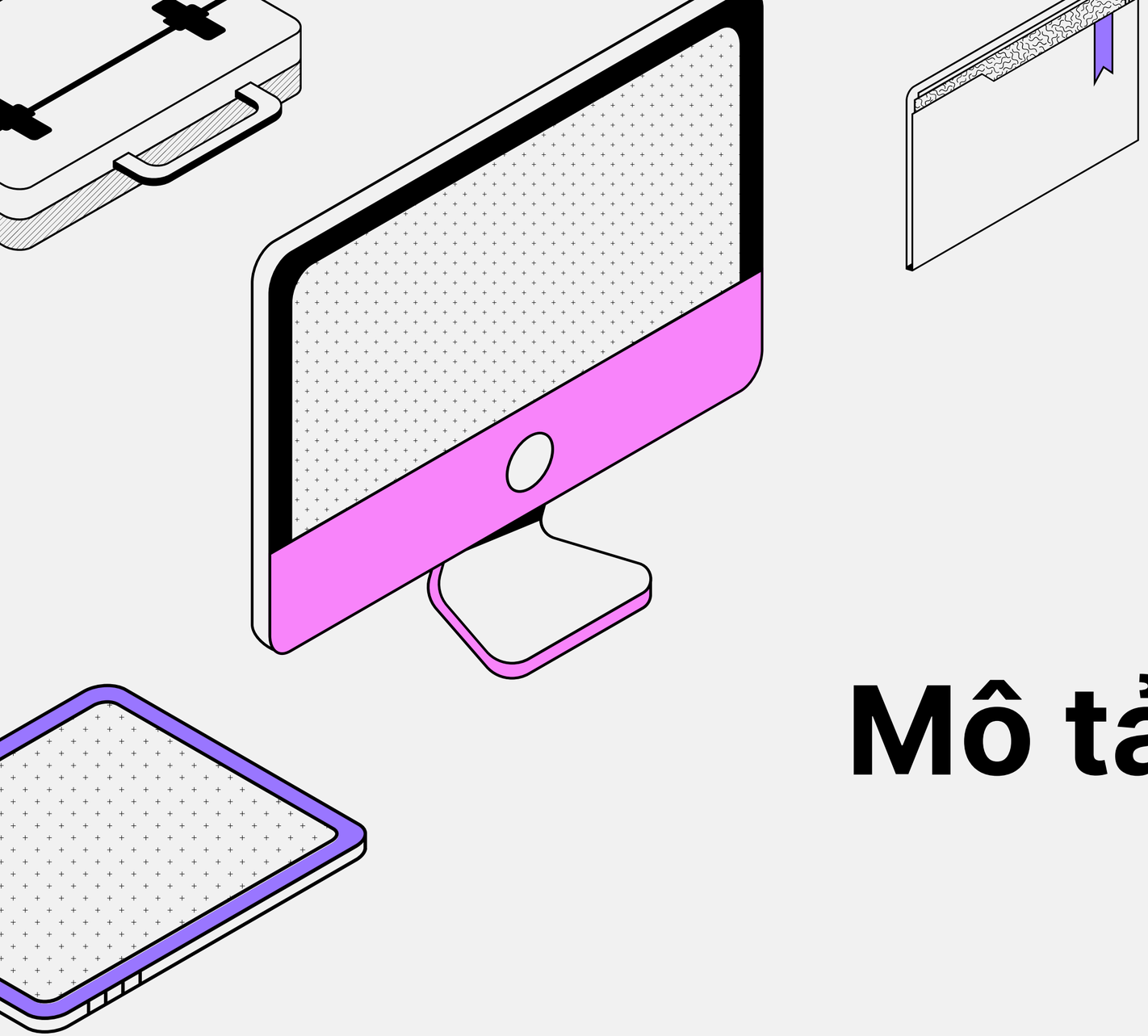
Trong môi trường mạng lưới hai chiều

- Các ô trống mà ta có thể di chuyển qua
- Các ô chứa vật cản mà ta không thể đi vào

Yêu cầu

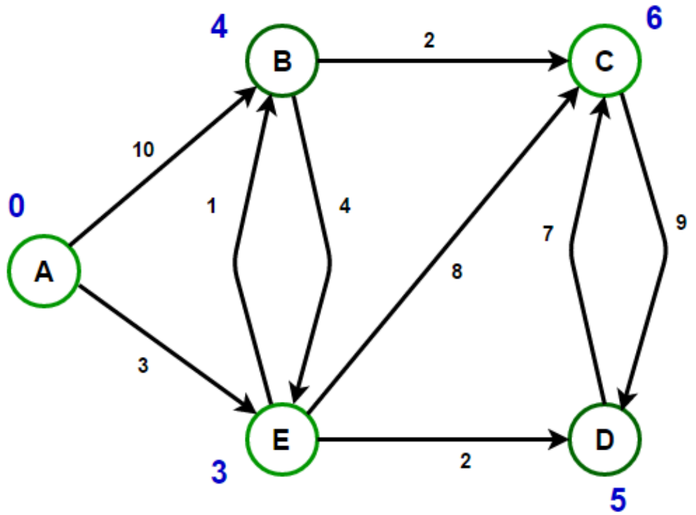
Tìm đường đi ngắn nhất

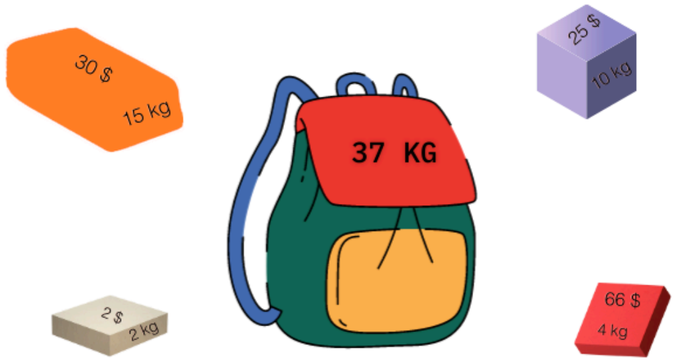
- Không đi xuyên qua vật cản
- Chi phí di chuyển là thấp nhất
- Nếu có nhiều đường đi, ưu tiên đường ngắn nhất



Mô tả thuật toán

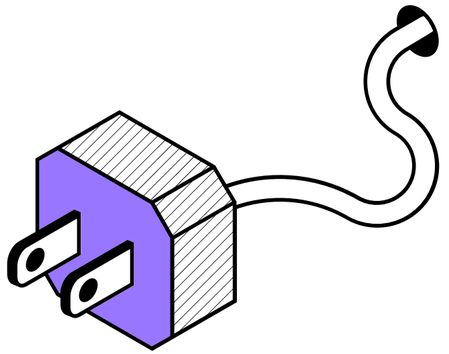
Mô tả thuật toán



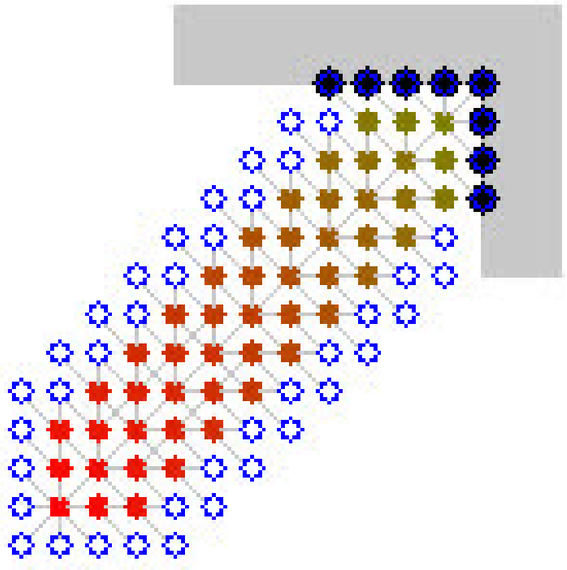


Thuật toán A* kết hợp các khía cạnh tốt nhất của thuật toán Dijkstra và thuật toán tìm kiếm tham lam

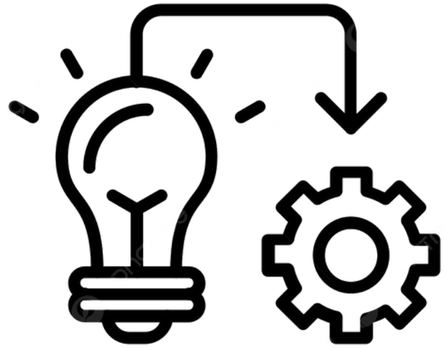
Cách hoạt động



A* mở rộng các nút từ điểm bắt đầu đến điểm kết thúc dựa trên hàm đánh giá $f(n)$. Hàm này kết hợp giữa chi phí đi từ điểm bắt đầu đến điểm hiện tại ($g(n)$) và ước lượng chi phí từ điểm hiện tại đến điểm đích ($h(n)$)



Cách thực hiện



Khởi tạo

- Đặt điểm bắt đầu vào danh sách mở (open list) và gán giá trị f ban đầu cho nó.
- Đặt danh sách đóng (closed list) rỗng.

Lặp lại

- Chọn nút trong danh sách mở có giá trị f nhỏ nhất và di chuyển nó sang danh sách đóng.
- Nếu nút này là điểm đích, thì đường đi ngắn nhất đã được tìm thấy.
- Nếu không, đối với mỗi nút kề cận:
  - Tính giá trị g và f cho nút kề.
  - Nếu nút này chưa có trong danh sách mở hoặc có giá trị f nhỏ hơn giá trị hiện tại trong danh sách mở, thì cập nhật giá trị và đặt nút này vào danh sách mở.

Kết thúc

Lặp lại quá trình cho đến khi tìm thấy điểm đích hoặc danh sách mở rỗng (đường đi không tồn tại).

Mã giả

A* (Q, S, G, P, c, h)

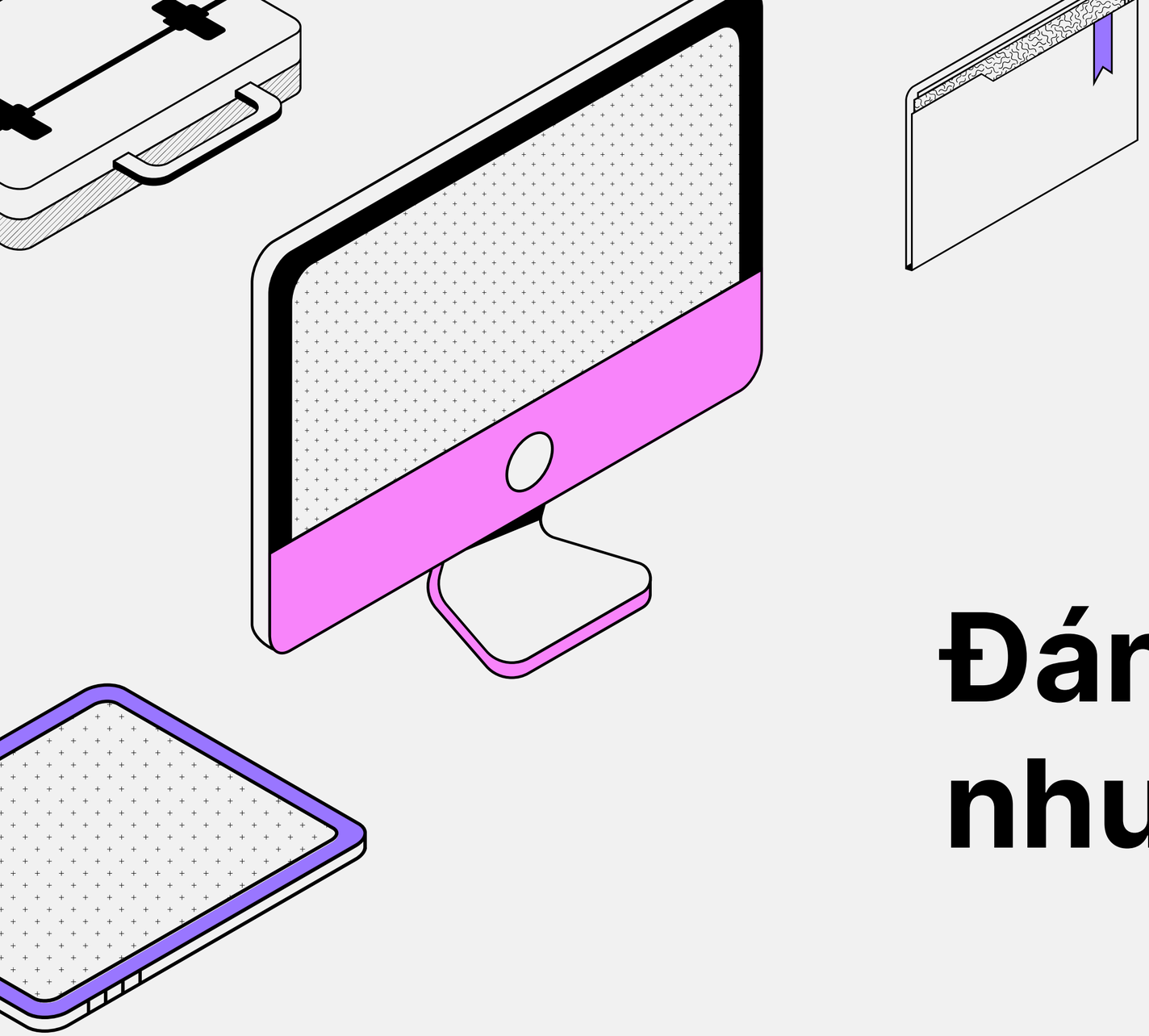
Đầu vào: Bài toán tìm kiếm, hàm heuristic h

Đầu ra: Trạng thái đích

Khởi tạo: O ← S (O: danh sách các nút mở)

while(O Ø) do:
    1. Lấy nút n có f (n) là nhỏ nhất khỏi O
    2. if n ∈ G, return (đường đi tới n)
    3. Với mọi m ∈ P(n):
        a. g(m) = g(n) + c(n, m)
        b. f (m) = g(m) + h(m)
        c. thêm m vào O cùng giá trị f (m)
return: Không tìm được đường đi.

```python
def a_star(start, goal, heuristic, get_neighbors, distance):
    # Khởi tạo danh sách mở và danh sách đóng
    open_list = [start]     # Các nút cần được đánh giá
    closed_list = set()     # Các nút đã đánh giá

    # Khởi tạo các thuộc tính của nút
    start.g = 0                         # Chi phí từ điểm bắt đầu đến chính nó là 0
    start.h = heuristic(start, goal)    # Ước lượng đến đích
    start.f = start.g + start.h         # Tổng chi phí ước tính
    start.parent = None                 # Để tái tạo đường đi

    while open_list:
        # Lấy nút có giá trị f thấp nhất - thực hiện bằng cách sử dụng hàng đợi ưu tiên
        # Để truy xuất nút tốt nhất nhanh hơn
        current = min(open_list, key=lambda node: node.f)

        # Kiểm tra xem chúng ta đã đến đích chưa
        if current == goal:
            return reconstruct_path(current)

        open_list.remove(current)
        closed_list.add(current)

        for neighbor in get_neighbors(current):
            if neighbor in closed_list:
                continue # Bỏ qua các nút đã được đánh giá

            # Tính toán điểm g tạm thời
            tentative_g = current.g + distance(current, neighbor)

            if neighbor not in open_list:
                open_list.append(neighbor)
            elif tentative_g >= getattr(neighbor, 'g', float('inf')):
                continue # Đường đi này không tốt hơn

            # Đây là đường đi tốt nhất cho đến thời điểm hiện tại
            neighbor.parent = current
            neighbor.g = tentative_g
            neighbor.h = heuristic(neighbor, goal)
            neighbor.f = neighbor.g + neighbor.h

    return None  # Không tồn tại đường đi

def reconstruct_path(current):
    path = []
    while current:
        path.insert(0, current)
        current = current.parent
    return path
```



Đánh giá ưu nhược điểm

Ưu điểm

Hiệu quả: A* thường rất nhanh

Hiệu quả cao: Sử dụng hàm đánh giá (f(n)) để ưu tiên các nút có khả năng dẫn đến đích cao hơn

Tính linh hoạt: Có thể điều chỉnh bằng cách thay đổi hàm heuristic

Nhược điểm

Tiêu tốn bộ nhớ: A* lưu trữ nhiều nút trong danh sách mở

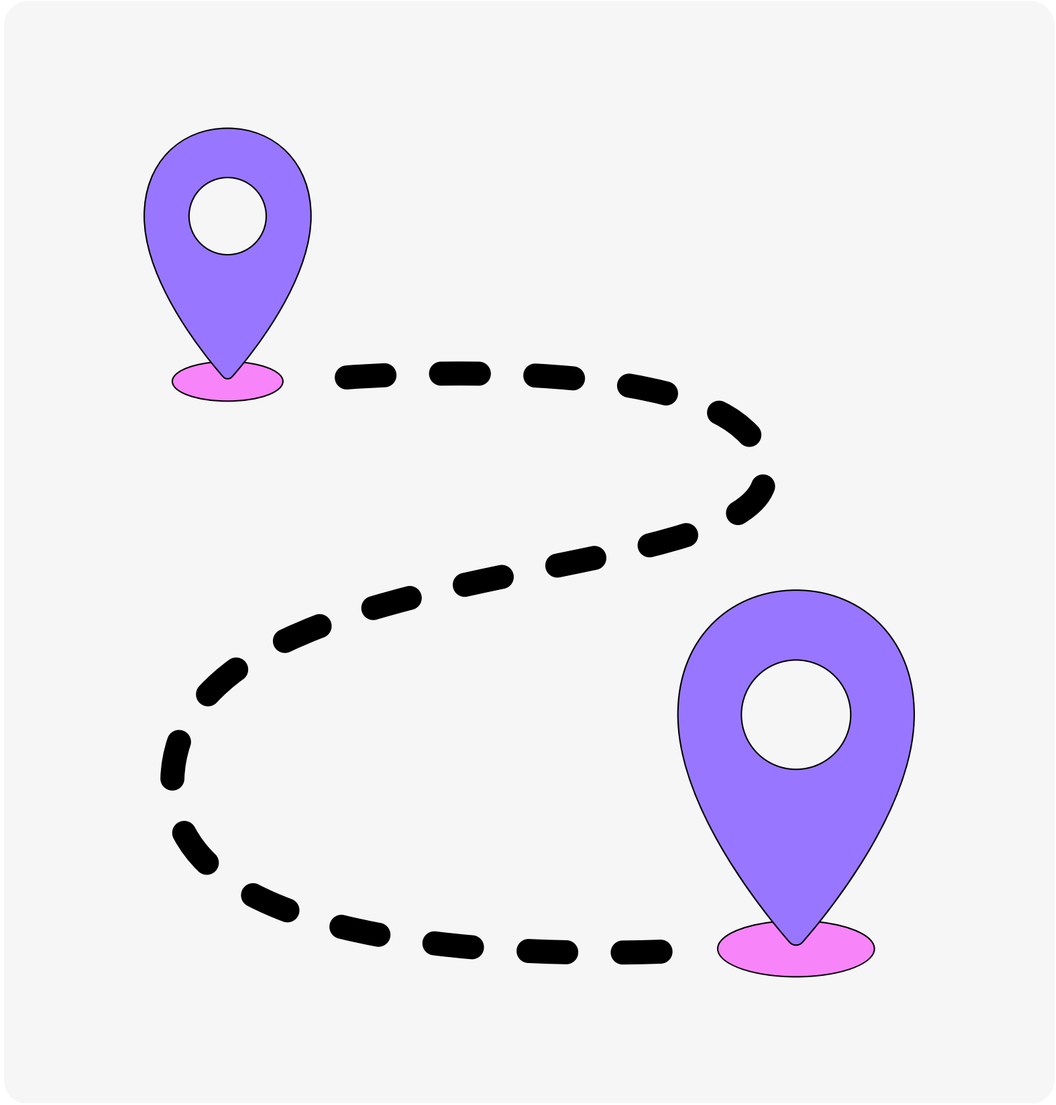
Phụ thuộc Heuristic: A* phụ thuộc vào chất lượng của hàm heuristic

So sánh và ứng dụng của thuật toán A* trong tìm đường đi ngắn nhất

So sánh A* với các thuật toán tìm đường khác

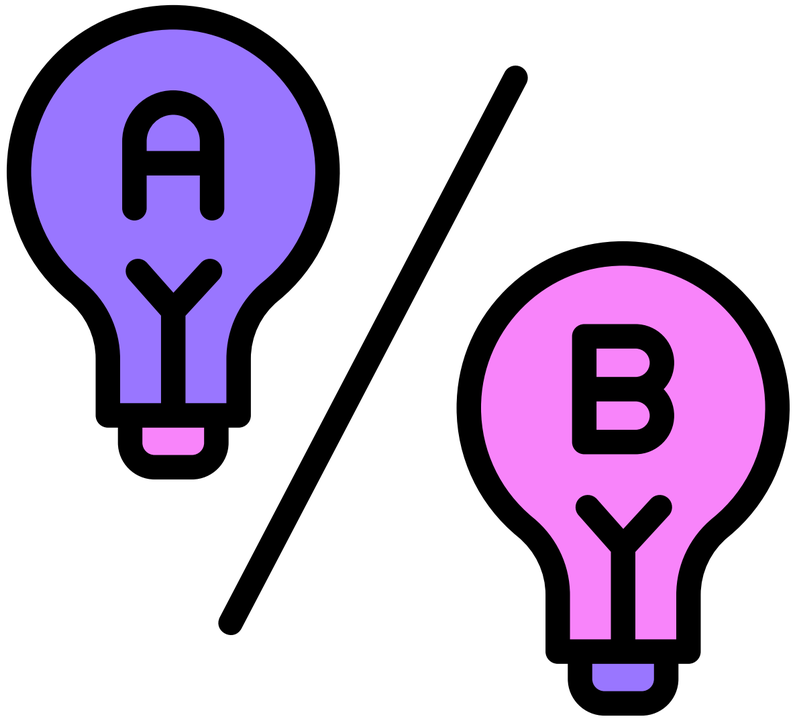
So sánh A* với Dijkstra

So sánh A* với BFS, DFS



So sánh A* và Dijkstra

- Cơ chế hoạt động
- Hiệu suất
- Tính tối ưu
- Độ phức tạp thời gian

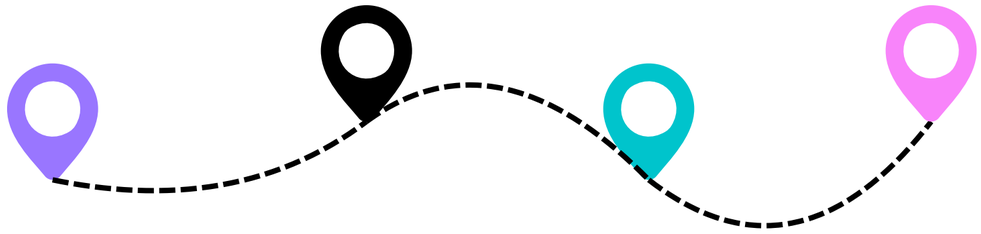


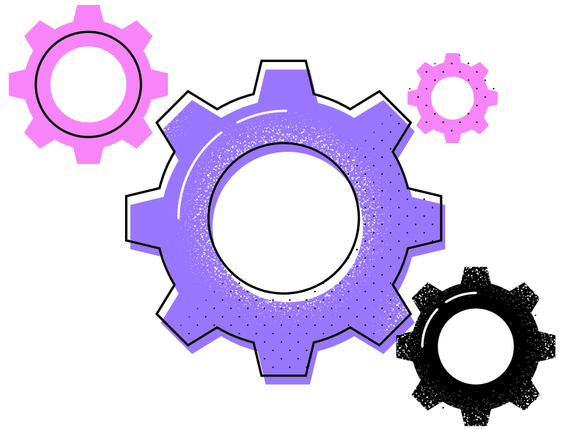
So sánh A* và Dijkstra

Cơ chế hoạt động

Dijkstra

- Thuật toán này tìm đường đi ngắn nhất từ một nút nguồn đến tất cả các nút khác trong đồ thị
- Nó hoạt động bằng cách luôn chọn nút có chi phí thực tế (g(n) - khoảng cách từ điểm bắt đầu đến nút hiện tại) nhỏ nhất để mở rộng
- Dijkstra không sử dụng thông tin ước lượng về đích đến





A*

- A* được coi là một phần mở rộng của thuật toán Dijkstra
- Nó cũng sử dụng chi phí thực tế g(n) nhưng kết hợp thêm hàm heuristic h(n) để ước lượng chi phí từ nút hiện tại đến nút đích
- Hàm đánh giá của A*: f(n) = g(n) + h(n)
- Việc sử dụng heuristic giúp A* hướng việc tìm kiếm về phía mục tiêu, thay vì mở rộng đều ra các hướng như Dijkstra

So sánh A* và Dijkstra

Hiệu suất

Dijkstra

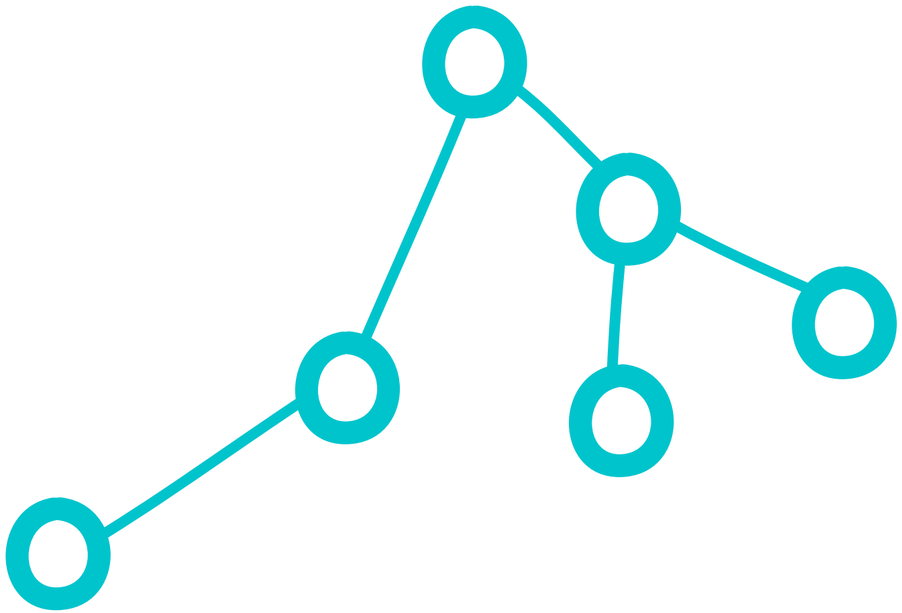
- Mặc dù đảm bảo tìm ra đường đi ngắn nhất, Dijkstra có thể tốn nhiều thời gian hơn khi khám phá các nút không cần thiết, đặc biệt trong các đồ thị lớn

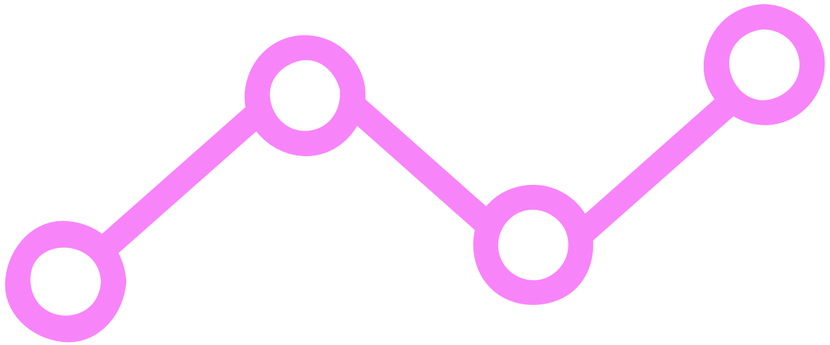
A*

- Nhờ có hàm heuristic, A* thường có hiệu suất cao hơn Dijkstra
- Nó có khả năng "nhìn trước" và xem xét các con đường có tiềm năng tốt hơn, giúp giảm thời gian tìm kiếm
- Nếu hàm heuristic được thiết kế hợp lý, A* có thể tìm ra đường đi tối ưu trong thời gian rất ngắn

So sánh A* và Dijkstra

Tính tối ưu





Dijkstra

- Đảm bảo tìm ra đường đi ngắn nhất nếu trọng số các cạnh không âm

A*

- Đảm bảo tìm ra đường đi ngắn nhất nếu hàm heuristic h(n) là "admissible"

So sánh A* và Dijkstra

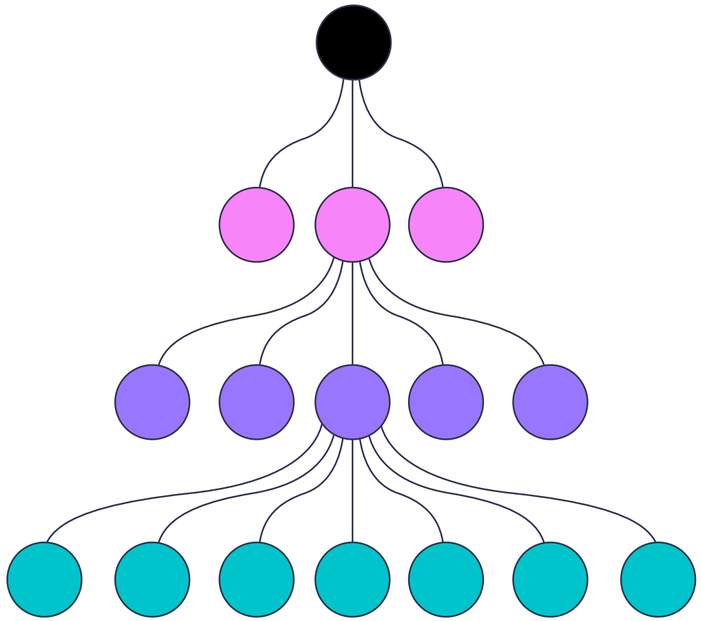
Độ phức tạp thời gian

Dijkstra

- Danh sách kề, min-heap cho hàng đợi ưu tiên, độ phức tạp là O((V + E) log V)
- Ma trận kề và duyệt mảng, độ phức tạp là $O(V^2)$
- Fibonacci heap cho hàng đợi ưu tiên, độ phức tạp có thể cải thiện xuống còn O(E + V log V)
- Độ phức tạp của Dijkstra trên lưới có thể coi là gần O(N log N) hoặc $O(N^2)$ tùy cách triển khai, với N là tổng số ô

A*

- Trường hợp xấu nhất: độ phức tạp thời gian có thể là hàm mũ.
- Trường hợp tốt: độ phức tạp thời gian của A* có thể là đa thức
- Độ phức tạp không gian: trong trường hợp xấu nhất cũng có thể là hàm mũ



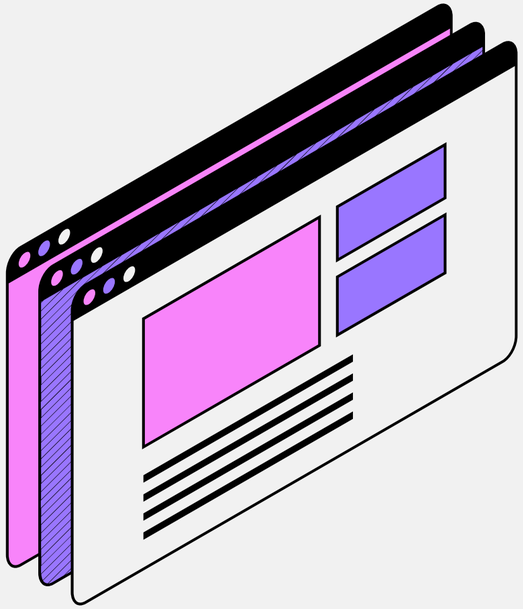
So sánh A* và Dijkstra

Độ phức tạp thời gian

Đặc điểm	Thuật toán Dijkstra	Thuật toán A*
Độ phức tạp thời gian (Worst Case)	$O(V^2)$ (với ma trận kề) hoặc $O(E \log V)$ (với danh sách kề và min-heap) hoặc $O(E + V \log V)$ (với Fibonacci heap)	Có thể là hàm mũ của độ dài lời giải nếu heuristic không tốt
Độ phức tạp thời gian (Typical/Good Heuristic)	Không áp dụng (Dijkstra không dùng heuristic)	Thường là đa thức và nhanh hơn Dijkstra đáng kể trong thực tế do heuristic hướng việc tìm kiếm
Độ phức tạp không gian (Worst Case)	$O(V)$ hoặc $O(V + E)$ tùy cách triển khai	Có thể là hàm mũ do phải lưu trữ nhiều nút trong danh sách Mở
Yếu tố ảnh hưởng chính	Số đỉnh (V), số cạnh (E), và cấu trúc dữ liệu sử dụng	Chất lượng của hàm heuristic h(n)

So sánh A* với BFS, DFS

Thuật toán A*



A* có khả năng định hướng tìm kiếm tốt hơn nhờ vào hàm heuristic

Thay vì duyệt tất cả các nút một cách mù quáng, A* ưu tiên các nút có khả năng dẫn đến đích cao hơn

A* tránh được nhược điểm của các thuật toán tìm kiếm theo best-first search thuần túy, vì nó không chỉ dựa vào heuristic mà còn tính đến chi phí thực tế từ điểm xuất phát

Ứng dụng của thuật toán A*

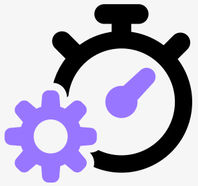
- Robot tự hành trong môi trường thực tế
- Điều hướng trong trò chơi điện tử (Game AI)
- Hệ thống định vị và dẫn đường trên mạng lưới đường bộ
- Lập kế hoạch đường đi cho robot di động trong môi trường phức tạp
- Định tuyến vận tải trong môi trường có vật cản động và tĩnh

Ứng dụng của thuật toán A*

Robot tự hành trong môi trường thực tế

Mô tả bài toán

Robot cần di chuyển từ điểm A đến điểm B trong môi trường vật lý và phải tự khám phá môi trường để tìm đường



Cách A* được dùng

Chọn nút tiếp theo cần mở rộng và một tầng thấp hơn để điều khiển sự di chuyển vật lý của robot trong môi trường

Ví dụ thực tế

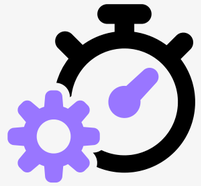
Robot dọn dẹp trong nhà, robot thăm dò trong các khu vực nguy hiểm, xe tự hành trong khuôn viên cần khám phá và định vị

Ứng dụng của thuật toán A*

Điều hướng trong trò chơi điện tử (Game AI)

Mô tả bài toán

Các NPC trong game cần di chuyển một cách thông minh qua các bản đồ 2D có nhiều vật cản



Cách A* được dùng

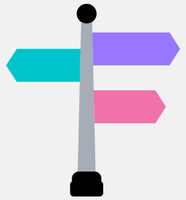
Bản đồ game được biểu diễn dưới dạng lưới ô vuông, A* được sử dụng để lập kế hoạch đường đi trên các lưới này

Ví dụ thực tế

Lính canh trong game chiến thuật, quái vật trong game nhập vai, các đơn vị quân sự di chuyển qua bản đồ chứa địa hình phức tạp

Ứng dụng của thuật toán A*

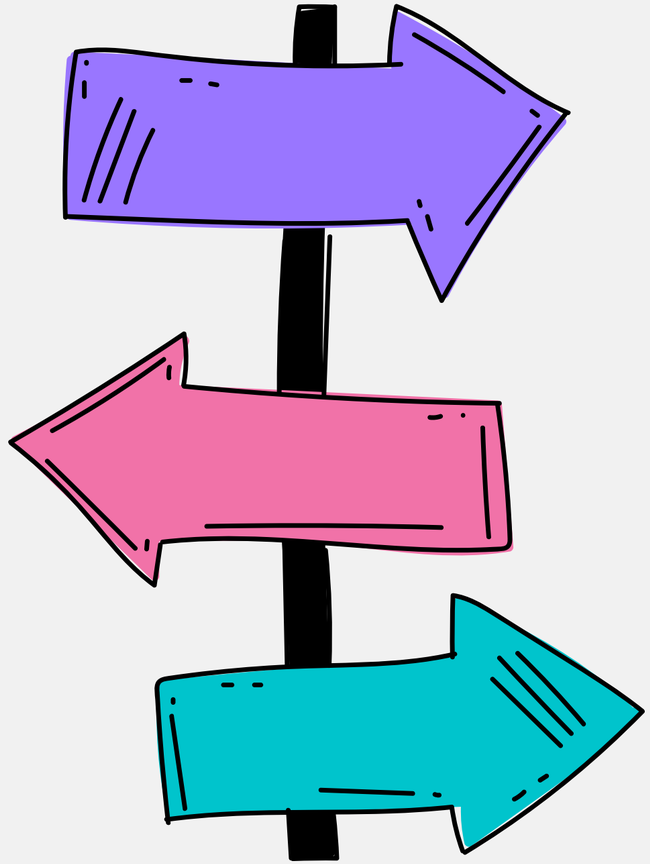
Hệ thống định vị và dẫn đường trên mạng lưới đường bộ



Các hệ thống định vị hiện đại cần tìm đường đi ngắn hoặc nhanh nhất giữa hai điểm trên bản đồ

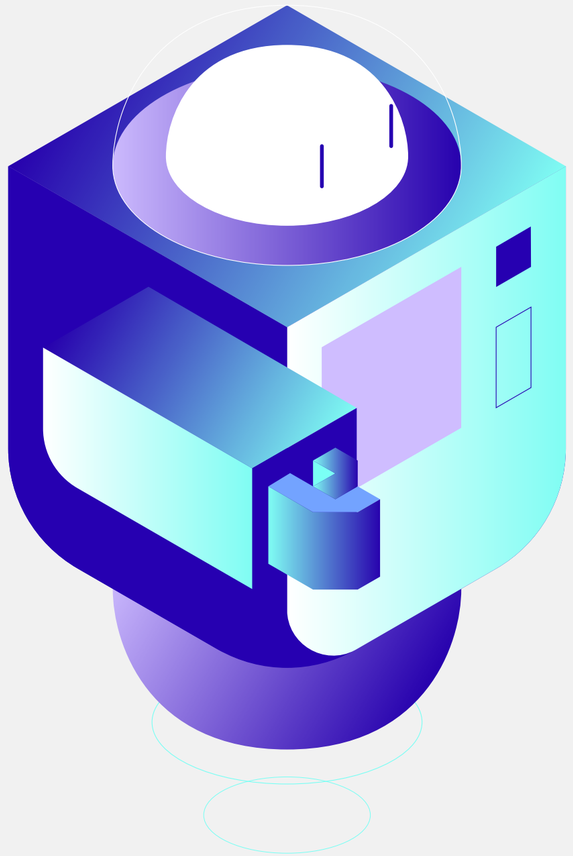
Áp dụng cho các mạng lưới đường bộ quy mô lớn, nhằm cải thiện hiệu quả bằng cách tích hợp các hàm heuristic mới, như "k-step look-ahead", để ước tính chi phí tốt hơn và giảm thời gian tính toán

Ứng dụng Google Maps hoặc các hệ thống GPS trên ô tô tính toán lộ trình di chuyển, có thể cập nhật theo tình hình giao thông thực tế



Ứng dụng của thuật toán A*

Robot di chuyển



Robot di động cần di chuyển trong các không gian có nhiều vật cản

Các nghiên cứu mở rộng A* để giải quyết các bài toán tối ưu đa mục tiêu hoặc để xử lý các lối đi hẹp bằng cách kết hợp với các kỹ thuật biểu diễn không gian khác như superquadrics và biểu đồ Voronoi để đảm bảo khoảng cách an toàn tối đa với vật cản

Ví dụ, Robot vận chuyển hàng hóa trong nhà kho có nhiều kệ hàng và lối đi hẹp, robot dịch vụ di chuyển trong các tòa nhà văn phòng

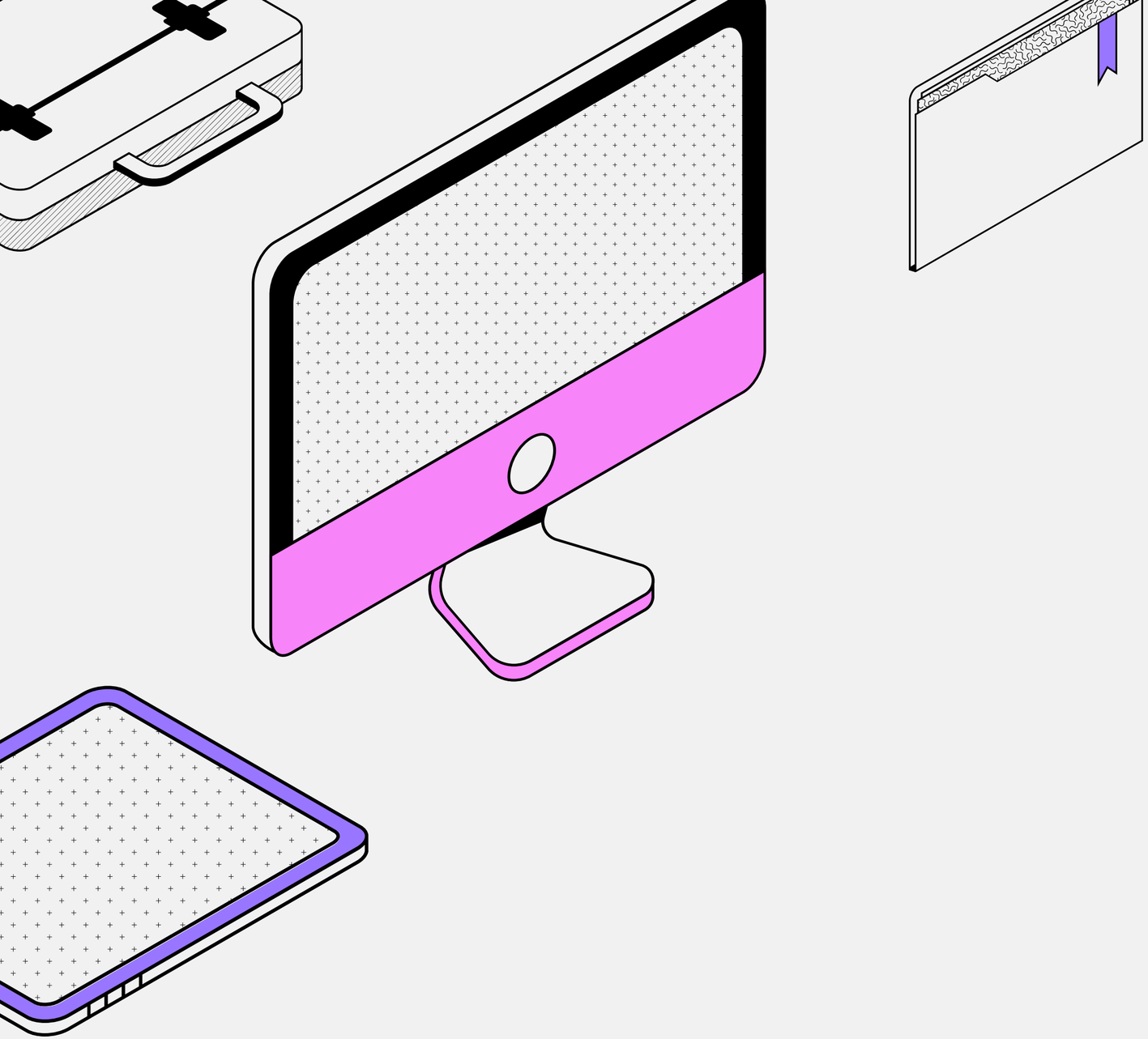
Ứng dụng của thuật toán A*

Định tuyến vận tải trong môi trường có vật cản động và tĩnh

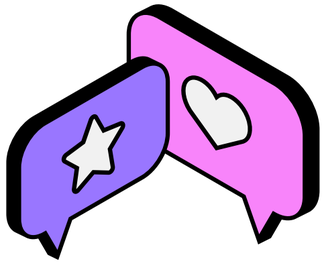
Tìm đường đi tối ưu cho các phương tiện vận tải trong môi trường có cả vật cản cố định và vật cản di chuyển là một bài toán thực tế quan trọng

Tập trung tối thiểu hóa thời gian giao hàng thay vì chỉ khoảng cách di chuyển, cùng bối cảnh tìm đường tối ưu trong môi trường có vật cản thay đổi, nơi các nguyên tắc của A* về đánh giá chi phí heuristic có thể được áp dụng hoặc so sánh

Xe tải giao hàng tự động điều chỉnh lộ trình dựa trên tình hình giao thông và các sự kiện bất ngờ



Demo



print('Thank you!')