ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

----- 🕮 ----

fit@hcmus
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN

Báo Cáo

ARES'S ADVENTURE

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

[Giáo viên lý thuyết]
GV. Lê Nhựt Nam

[Sinh viên báo cáo]
23127116 – Nguyễn Quang Thái

23127254 – Nguyễn Thị Như Quỳnh

23127366 – Võ Lê Ngọc Hiếu

23127524 – Hình Diễm Xuân

THÀNH PHỐ HỒ CHÍ MINH 2025 – 2026

Nội dung

I. Thông tin chung

1. Thông tin thành viên

MSSV	Họ tên	Email	Số điện thoại
23127116	Nguyễn Quang Thái	nqthai23@clc.fitus.edu.vn	0837848237
23127254	Nguyễn Thị Như Quỳnh	ntnquynh23@clc.fitus.edu.vn	0963745770
23127366	Võ Lê Ngọc Hiếu	vlnhieu23@clc.fitus.edu.vn	0792421305
23127524	Hình Diễm Xuân	hdxuan23@clc.fitus.edu.vn	0396723858

2. Bảng phân công công việc

MSSV	Công việc	Mức độ hoàn thành
23127116	-Giao diện -Thuật toán BFS -Phân tích và giải thích testcase	Tốt
23127254	-Thuật toán GBFS -Nội dung của các thuật toán -So sánh kết quả các thuật toán -Báo cáo	Tốt
23127366	-Thuật toán A* -Thuật toán UCS -So sánh kết quả các thuật toán	Tốt
23127524	-Giao diện -Thuật toán DFS -Phân tích và giải thích code giao diện -Phân tích và giải thích testcase	Tốt

- Bảng trên liệt kê các thành viên chịu trách nhiệm chính cho từng nhiệm vụ cụ thể trong đồ án. Tuy nhiên trong quá trình làm việc, mọi người luôn sẵn sàng hỗ trợ lẫn nhau, cùng nhau thảo luận, đóng góp ý kiến và chia sẻ kinh nghiệm để giải quyết những vấn đề phát sinh cũng như hoàn thiện đồ án.

3. Đánh giá

Criteria	Scores
Implement BFS correctly	1
Implement DFS correctly	1
Implement UCS correctly	1
Implement A* correctly	1
Implement GBFS correctly	1
Dijkstra's algorithm	1
Generate at least 10 test cases for each level with different at-tributes.	1.5
Result (output file and GUI).	1
Videos to demonstrate all algorithms for some test cases.	1
Report your algorithm and experiment with some reflection or com-ments.	2
TOTAL	11.5

II. Nội dung chi tiết

1. Hàm giao diện

+def draw_board(screen, title):

Vẽ nền màu xám nhạt.

Gọi hàm viết tiêu đề.

Vẽ các nút chọn cấp độ trò chơi.

Vẽ các nút bấm thuật toán.

Duyệt từng ô của ma trận.

Xác định vị trí các chủ thể.

Vẽ các ô màu tương ứng với các chủ thể như đã quy định.

Hiển thị trọng số của đá.

Cập nhật màn hình.

+def animate_solution(screen, solution, title):

Chạy vòng lặp cho đến khi đọc hết đường đi.

Khi người dùng đóng cửa sổ (bằng chuột) sẽ đóng trò chơi.

Khi người dùng bấm chuột vào nút “pause” hay “reset” thì thực hiện, bấm “pause” lần thứ 2 thì chương trình tiếp tục chạy.

Cập nhật bước đi tiếp theo từ “solution”.

Tính vị trí tiếp theo của nhân vật .

Kiểm tra tính hợp lệ của vị trí mới.

Nếu ô tiếp theo là đá:

Cập nhật vị trí đá.

Kiểm tra xem đá có thể di chuyển không.

Cập nhật trạng thái đá và đích đến nếu có thay đổi.

Cập nhật vị trí của nhân vật.

Cập nhật bảng.

Thời gian tạm dừng là 0.1 giây để tạo hiệu ứng chuyển động.

Sau khi chạy xong in thông báo đã hoàn thành.

+def draw_buttons(screen):

Tạo danh sách các nút ghi tên các thuật toán.

Vẽ từng nút lên màn hình.

Trả về danh sách vị trí các nút đã vẽ.

+def draw_title(screen, tile_size, in_title):

Xác định vị trí tiêu đề (căn lề phải).

Hiển thị các ký tự lên màn hình.

+def draw_level(screen):

Tạo danh sách các nút ghi tên các cấp độ của trò chơi.

Vẽ các nút theo hàng dọc, bên phải của màn hình.

Trả về danh sách vị trí các nút đã vẽ.

+def main():

Khởi tạo pygame và cửa sổ hiển thị trò chơi.

Khởi tạo các biến để điều phối trò chơi.

Vòng lặp đến khi ngừng chạy (running = false).

Tô màu màn hình.

Vẽ nút bấm các thuật toán.

Nếu ở trạng thái "menu" thì vẽ nút bấm thuật toán và cấp độ.

Nếu ở trạng thái chạy (running):

Thực hiện thuật toán tương ứng với các nút bấm.

Quay lại trạng thái "menu" sau khi chạy xong.

Cập nhật màn hình.

Xử lý các sự kiện:

Nếu người dùng đóng cửa sổ thì thoát trò chơi.

Xác định người dùng bấm chuột vào nút nào để chạy trò chơi theo yêu cầu tương ứng.

Thoát pygame.

2. Breadth-First Search (BFS)

2.1. Khái niệm

- BFS (Breadth-First Search) là thuật toán tìm kiếm theo chiều rộng trong đồ thị hoặc cây. Nó duyệt từng lớp (level) của đồ thị trước khi đi sâu hơn, đảm bảo rằng tất cả các đỉnh ở mức gần hơn được thăm trước.

2.2. Đặc điểm

-Tìm đường đi ngắn nhất trong đồ thị không trọng số.

-Không bị rơi vào vòng lặp vô hạn nếu dùng danh sách đánh dấu.

-Tốt cho tìm kiếm theo lớp, kiểm tra đồ thị liên thông.

-Tốn bộ nhớ hơn DFS, vì phải lưu nhiều đỉnh cùng lúc trong hàng đợi.

-Chậm hơn DFS với đồ thị có nhiều nhánh và lời giải nằm sâu.

2.3. Mã giả

```
Input: rootVertex
Result: Path to a solution vertex
Begin
visited ← ∅  // Set of visited vertices
visited.add(rootVertex)
queue.push(rootVertex)
```

```
while pq ≠ ∅ do
        vertex = queue.pop()
        visited.add(vertex)
        foreach successor of vertex do
                if stone in deadlock square then continue
                if successor not in visited then
                        if isSolution(vertex) then return pathTo(vertex)
                        queue.push(successor)
return no solution found.
End
```

3. Depth-First Search (DFS)

3.1. Khái niệm

- DFS (Depth-First Search) là thuật toán tìm kiếm theo chiều sâu trong đồ thị hoặc cây. Nó duyệt từ một đỉnh ban đầu, sau đó đi sâu vào các đỉnh con trước khi quay lại đỉnh cha.

3.2. Đặc điểm

-Bộ nhớ thấp hơn so với BFS, vì chỉ lưu trữ đường đi hiện tại.

-Hiệu quả với đồ thị có nhánh sâu, đặc biệt khi lời giải nằm sâu.

-Dễ triển khai bằng đệ quy, giúp viết mã đơn giản hơn.

-Không đảm bảo tìm được đường đi ngắn nhất, vì có thể đi vào nhánh sai trước.

-Có thể bị rơi vào vòng lặp vô hạn, nếu không đánh dấu các nút đã thăm trong đồ thị có chu trình.

-Không tối ưu trong tìm kiếm trạng thái tối ưu, vì không xét chi phí đường đi.

3.3. Mã giả

```
Input: rootVertex
Result: Path to a solution vertex
Begin
visited ← ∅  // Set of visited vertices
visited.add(rootVertex)
stack.push(rootVertex)
while pq ≠ ∅ do
        vertex = stack.pop()
        visited.add(vertex)
        foreach successor of vertex do
                if stone in deadlock square then continue
                if successor not in visited then
                        if isSolution(vertex) then return pathTo(vertex)
```

stack.push(successor)

return no solution found.

End

4. **Uniform Cost Search (UCS)**

4.1. Khái niệm

-UCS (Uniform Cost Search) là thuật toán tìm kiếm không có heuristic, dựa hoàn toàn vào chi phí đường đi thực tế để tìm ra giải pháp tối ưu.

4.2. Đặc điểm

-Đảm bảo tìm đường đi tối ưu nếu tất cả chi phí đều không âm.

-Không bị rơi vào bẫy cục bộ, vì luôn chọn nút có chi phí nhỏ nhất.

-Hoạt động tốt với đồ thị có trọng số, đặc biệt là khi có nhiều đường đi với chi phí khác nhau.

-Có thể mở rộng nhiều nút, dẫn đến tiêu tốn bộ nhớ nếu đồ thị quá lớn.

-Chạy chậm nếu có quá nhiều trạng thái, vì phải kiểm tra tất cả các đường có chi phí thấp nhất trước khi tiếp tục.

4.3. Mã giả

Input: rootVertex

Result: Path to a solution vertex

Begin

visited ← ∅ // Set of visited vertices

visited.add(rootVertex)

queue.push(rootVertex)

while pq ≠ ∅ **do**

vertex = queue.pop()

visited.add(vertex)

foreach *successor of vertex* **do**

if stone in deadlock square **then continue**

if successor not in visited **then**

if isSolution(vertex) **then** return pathTo(vertex)

queue.push(successor)

return *no solution found.*

End

4.4. Quá trình thực hiện

Bước 1: Khởi tạo hàng đợi ưu tiên (dựa trên chi phí) với nút xuất phát (chi phí = 0)

Bước 2: Khởi tạo vòng lặp cho đến khi có được solution hoặc hàng đợi ưu tiên rỗng

+Lấy nút có chi phí thấp nhất ra khỏi hàng đợi

+Kiểm tra xem nút đó có phải là đích hay không. Nếu có, trả về solution

+Nếu không, thêm nút vào visited

+Mở rộng các nút con từ nút hiện tại

- Nếu nút con hiện tại là "dead lock", bỏ qua nút này
- Nếu không, kiểm tra xem nút con có tồn tại trong visited hay chưa hoặc chi phí mới của nút con có tối ưu hơn hay không
- Nếu có, thêm nút con với chi phí mới vào hàng đợi ưu tiên

4.5. Hàm giá trị

f(n) = g(n). Trong đó, g(n) là chi phí di chuyển + đẩy đá (nếu có)

5. A* Search with heuristic

5.1. Khái niệm

-A* là một trong những thuật toán Informed Search tốt nhất và phổ biến nhất trong các bài toán tìm kiếm đường đi. A* tìm đường ngắn nhất từ nút xuất phát đến mục tiêu bằng cách sử dụng thêm heuristic vào hàm tính toán của mình.

5.2. Đặc điểm

-Tối ưu & hoàn chỉnh: Nếu heuristic h(n)h(n)h(n) là **admissible** (không đánh giá quá cao) và **consistent** (thỏa mãn điều kiện tam giác), A* sẽ tìm được đường đi ngắn nhất.

-Tốc độ phụ thuộc vào heuristic: Nếu heuristic tốt, A* có thể rất nhanh. Nếu h(n) = 0, A* trở thành UCS (Dijkstra).

5.3. Mã giả

Input: rootVertex

Result: Path to a solution vertex

Begin

visited ← ∅ // Set of visited vertices

visited.add(rootVertex)

queue.push(rootVertex)

while pq ≠ ∅ **do**

    vertex = queue.pop()

    visited.add(vertex)

foreach *successor of vertex* **do**

if stone in deadlock square **then continue**

if successor not in visited **then**

if isSolution(vertex) **then** return pathTo(vertex)

queue.push(successor)

return *no solution found.*

End

5.4. Quá trình thực hiện

Bước 1: Khởi tạo hàng đợi ưu tiên (dựa trên chi phí) với nút xuất phát (chi phí = 0)

Bước 2: Khởi tạo vòng lặp cho đến khi có được solution hoặc hàng đợi ưu tiên rỗng

+Lấy nút có chi phí thấp nhất ra khỏi hàng đợi

+Kiểm tra xem nút đó có phải là đích hay không. Nếu có, trả về solution

+Nếu không, thêm nút vào visited

+Mở rộng các nút con từ nút hiện tại

- Nếu nút con hiện tại là "dead lock", bỏ qua nút này
- Nếu không, kiểm tra xem nút con có tồn tại trong visited hay chưa hoặc chi phí mới của nút con có tối ưu hơn hay không
- Nếu có, thêm nút con với chi phí mới vào hàng đợi ưu tiên

5.5. Hàm giá trị

$$f(n) = g(n) + h(n)$$

```
def heuristic(stones):
    return sum(min(distances[switch][s.point] for switch in switches) for s in stones)
```

-Hàm heuristic hiện tại tính toán tổng khoảng cách từ các switch đến viên đá gần nhất tương ứng với mỗi switch. Với độ phức tạp O(1), việc dùng heuristic này có thể tạm chấp nhận được khi mà A* luôn tìm ra suing đi với chi phí tối ưu nhất.

6. Greedy Best-first Search (GBFS)

6.1. Khái niệm

-GBFS (Greedy Best-First Search – Tìm kiếm tham lam tốt nhất) là một thuật toán tìm kiếm heuristic trong trí tuệ nhân tạo. Nó hoạt động bằng cách luôn mở rộng nút có giá trị heuristic tốt nhất (tức là ước tính gần nhất đến mục tiêu) mà không quan tâm đến chi phí đã đi qua.

6.2. Đặc điểm

-Chiến lược: Chọn nút có giá trị heuristic thấp nhất h(n) để mở rộng.

-Không đảm bảo tối ưu: Vì không xét đến chi phí đường đi thực tế, GBFS có thể chọn một đường đi không tối ưu.

-Hiệu suất cao hơn so với tìm kiếm mù (như BFS hoặc DFS) khi heuristic tốt.

-Dễ bị mắc kẹt trong cục bộ tối ưu nếu heuristic không chính xác.

6.3. Mã giả

Input: rootVertex

Result: Path to a solution vertex

Begin

visited ← ∅ // Set of visited vertices

visited.add(rootVertex)

queue.push(rootVertex)

while pq ≠ ∅ **do**

vertex = queue.pop()

visited.add(vertex)

foreach *successor of vertex* **do**

if stone in deadlock square **then continue**

if successor not in visited **then**

if isSolution(vertex) **then** return pathTo(vertex)

queue.push(successor)

return *no solution found.*

End

6.4. Quá trình thực hiện

Bước 1: Khởi tạo hàng đợi ưu tiên (dựa trên chi phí) với nút xuất phát (chi phí = 0)

Bước 2: Khởi tạo vòng lặp cho đến khi có được solution hoặc hàng đợi ưu tiên rỗng

+Lấy nút có chi phí thấp nhất ra khỏi hàng đợi

+Kiểm tra xem nút đó có phải là đích hay không. Nếu có, trả về solution

+Nếu không, thêm nút vào visited

+Mở rộng các nút con từ nút hiện tại

- Nếu nút con hiện tại là “dead lock”, bỏ qua nút này
- Nếu không, kiểm tra xem nút con có tồn tại trong visited hay chưa hoặc chi phí mới của nút con có tối ưu hơn hay không
- Nếu có, thêm nút con với chi phí mới vào hàng đợi ưu tiên

6.5. Hàm giá trị

$$f(n) = h'(n)$$

```
new_cost = heuristic(new_stones) + stone_weight
```

-Trong đó h’(n) được tính bằng hàm heuristic của A* cộng với khối lượng đá tương ứng trong mỗi bước di chuyển, cụ thể stone_weight có thể có các giá trị:

+0 nếu nhân vật di chuyển tự do

+Khối lượng viên đá tương ứng trong mỗi bước di chuyển

-Hàm h’(n) này với chi phí O(1) đã cho ra đường đi có mức chi phí tối ưu với số steps có thể chấp nhận được.

7. Dijkstra’s algorithm

7.1. Khái niệm

- Dijkstra là một thuật toán tìm đường đi ngắn nhất trong đồ thị có trọng số không âm. Nó được đề xuất bởi nhà khoa học máy tính Edsger W. Dijkstra vào năm 1956.

7.2. Đặc điểm

- Tính chất tham lam: Luôn chọn đỉnh có khoảng cách tạm thời nhỏ nhất để mở rộng.

- Áp dụng cho trọng số không âm: Chỉ hoạt động đúng với đồ thị không chứa cạnh có trọng số âm.

- Cơ chế "relaxation": Liên tục cập nhật khoảng cách đến các đỉnh kề nếu tìm được đường đi ngắn hơn.

- Độ phức tạp:

+ Sử dụng danh sách kề + min-heap: O((V + E) log V)

+ Sử dụng ma trận kề: O(V^2)

7.3. Quá trình thực hiện

-Bước 1: Khởi tạo

+Gán khoảng cách từ đỉnh nguồn đến chính nó là 0 và các đỉnh khác là vô cùng

+ Khởi tạo một hàng đợi ưu tiên (min-heap) chứa các đỉnh cần xử lý.

-Bước 2: Lặp qua hàng đợi: Lấy đỉnh có khoảng cách tạm thời nhỏ nhất ra khỏi hàng đợi.

-Bước 3: Cập nhật (Relaxation): Với mỗi đỉnh kề của đỉnh vừa lấy, kiểm tra và cập nhật khoảng cách nếu tìm được đường đi ngắn hơn.

-Bước 4: Lặp lại: tiếp tục quá trình cho đến khi hàng đợi rỗng

-Bước 5: Kết quả: Mảng khoảng cách lưu lại đường đi ngắn nhất từ đỉnh nguồn tất cả các đỉnh khác

III. Mô tả và so sánh kết quả thuật toán

1. Mô tả testcase

-Level 1:

Kích thước 8x10

Số lượng đá 1 với khối lượng 1

Thử thách chỉ có 1 đường qua switch và ma trận rộng gây khó khăn với nhiều đường trống

-Level 2:

Kích thước 8x10

Số lượng đá 1 với khối lượng bằng 1

Thử thách ma trận rộng có nhiều đường giải yêu. Có thể thử thách tìm đường đi tối ưu cho các thuật toán

-Level 3:

Kích thước 8x8

Số lượng đá 1 khối lượng 1

Thử thách nhiều tường ở trong maze tạo thành nhiều đường đi khác nhau, độ phức tạp cũng tăng lên do có nhiều deadlocks

-Level 4:

Kích thước 8x8

2 đá với 2 khối lượng khác nhau.

Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng

-Level 5:

Kích thước 8x8

2 đá với 2 khối lượng khác nhau.

Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

-Level 6:

Kích thước 9x8

Số lượng đá 2 với 2 khối lượng khác nhau.

Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

-Level 7:

Kích thước 8x8

Số lượng đá 3 với 3 khối lượng khác nhau.

Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

-Level 8:

Kích thước 8x8

Số lượng đá 3 với 3 khối lượng khác nhau, nhiều vách tường hơn tạo độ khó.
Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

-Level 9:

Kích thước 8x7
Số lượng đá 3 với 3 khối lượng khác nhau, vách tường sắp xếp tạo độ khó hơn.
Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

-Level 10:

Kích thước 9x9
Số lượng đá 3 với 3 khối lượng khác nhau, tăng khối lượng lên cao, vách tường sắp xếp tạo độ khó hơn.
Thử thách các thuật toán trong việc tìm đường và hệ số liên quan đến khối lượng, cùng với nhiều deadlocks hơn

2. So sánh kết quả thuật toán

BFS					
Level	Node generated	Steps	Weight	Time (ms)	Memory (MB)
1	121	16	5	1.000165	0.007812
2	2435	23	10	9.931802	0.222656
3	816	37	6	1.440286	0.027343
4	3561	61	26	6.522178	0.398437
5	23729	82	150	43.649435	0.738281
6	25612	104	74	48.675060	0.289062
7	6888	70	32	14.845848	0.300781
8	134343	100	172	297.226667	5.851562
9	52242	84	103	119.625329	1.132812
10	743581	121	1413	1981.358	42.52734

DFS					
Level	Node generated	Steps	Weight	Time (ms)	Memory
1	217	22	5	1.515150	0.023437
2	1046	27	10	1.268864	0.109375
3	235	71	6	1.612663	0.031250
4	499	115	34	7.512331	0.0546875
5	4662	290	500	25.888442	0.609375
6	3973	508	266	9.490251	1.398437
7	920	174	72	3.073454	0.277344
8	8082	312	288	17.354272	1.207031
9	3336	274	257	8.30625	0.757812
10	247005	10211	40859	1648.355	265.0156

UCS					
Level	Node generated	Steps	Weight	Time (ms)	Memory (MB)
1	109	16	5	0.000001	0.015625
2	799	23	10	2.079964	0.121094
3	385	37	6	2.120256	0.070312
4	2308	61	26	6.095886	0.496094
5	6695	82	150	16.247988	1.394531
6	9101	104	74	22.317648	1.79687
7	5458	70	32	12.374163	1.308594
8	33142	106	178	93.628168	6.368719
9	16112	92	65	51.09977	3.691406
10	382115	121	1413	1492.5889	68.410156

A*					
Level	Node generated	Steps	Weight	Time (ms)	Memory
1	91	16	5	1e-06	0.011719
2	571	23	10	1.000404	0.082031
3	359	37	6	1.488447	0.054688
4	2063	63	26	12.566566	0.015625
5	6217	94	150	46.970367	0.70731
6	7107	104	74	22.384882	1.570312
7	5265	70	32	24.638414	1.359375
8	37220	100	172	179.776669	7.488281
9	14570	86	85	82.597017	3.710938
10	379301	133	1413	2284.645319	68.722656

GBFS					
Level	Node generated	Steps	Weight	Time (ms)	Memory
1	129	16	5	1.005173	0.019531
2	242	23	10	1.316786	0.042969
3	170	43	6	0.999212	0.027344
4	1427	87	28	4.999399	0.277344
5	1929	100	210	6.999731	0.367188
6	4245	158	96	15.803337	0.960938
7	4969	100	33	22.717476	1.199219
8	26639	194	186	125.565052	5.484375
9	1790	94	65	9.493113	0.449219
10	304656	605	1651	1614.073992	59.617188

- **Theo level**

-BFS:
+Node generated: tăng nhanh từ level 5
+Steps: không tăng đều mà có sự dao động
+Time: tăng dần, tăng đột biến từ level 7
+Memory: tăng đột biến từ level 7, level 10 tăng rất cao

-DFS:
+Node generated: tăng mạnh, bùng nổ ở level 10
+Steps: tăng dần, đặc biệt cao ở level 10
+Time: thời gian tăng dần theo level nhưng đột biến ở level 10
+Memory: tăng theo level nhưng level tiêu tốn tài nguyên nhiều nhất

-UCS:
+Node generated: tăng mạnh theo cấp số nhân
+Steps: tăng dần nhưng không theo quy luật chặt chẽ
+Time: tăng dần, tăng mạng từ level 4
+Memory: tăng dần nhưng có biến động

-A*:
+Node generated: tăng nhưng có một số mức giảm
+Steps: tăng nhưng không ổn định
+Time: tăng dần theo level, ngoại trừ một số trường
+Memory: tăng dần nhưng không hoàn toàn ổn định, tăng mạnh từ level 7 trở đi

-GBFS:
+Node Generated: số node tăng nhanh chóng, đặc biệt từ level 8 trở đi có sự nhảy vọt (từ 26,639 lên 304,656).
+Steps: số bước tăng đều nhưng chậm hơn so với số node được tạo.
+Time: từ level 8 trở đi, thời gian thực thi tăng vọt từ 125.56 ms lên hơn 1600 ms.
+Memory: tăng nhanh, cho thấy thuật toán cần nhiều tài nguyên hơn ở level cao.

- **Theo thuật toán:**

-Nhìn chung, BFS là thuật toán tốn thời gian nhất, node generated cao nhất, solution đưa ra thuộc mức ổn tuy chưa tối ưu.
-DFS có thời gian chạy thuật toán nhanh nhất, node generated thấp hơn so với các thuật toán còn lại, tuy vậy solution đưa ra là khá "đần".
-UCS, A* có thời gian chạy thuật toán ở mức trung bình, node generated thấp hơn tương dối so với bfs (Node generated của A* thường ít hơn UCS), cả hai cho ra solution tối ưu nhất trong số 5 thuật toán

-Thời gian chạy thuật toán của GBFS và node generated chỉ cao hơn so với DFS. Tuy vậy, solution GBFS sinh ra ở mức "chấp nhận được".

IV. Video link

-Demo Video: https://byvn.net/1lvm

V. Tham khảo

-Github: https://short.com.vn/a8XU
-ChatGPT
-Youtube