

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO TÓM TẮT
ĐỒ ÁN SỐ 01
CÀI ĐẶT & SO SÁNH CÁC THUẬT TOÁN TÌM KIẾM

Học phần: Cơ sở trí tuệ nhân tạo

Lớp: CQ2019/22

Họ và tên các thành viên:

- 1. Bùi Lê Tuấn Anh – 19120163**
- 2. Ngô Nhật Tân – 19120128**

Thành phố Hồ Chí Minh, tháng 11 năm 2021

Nội dung

1. CÁC THUẬT TOÁN TÌM KIẾM	1
2. BẢN ĐỒ KHÔNG CÓ ĐIỂM THƯỜNG	6
3. BẢN ĐỒ CÓ ĐIỂM THƯỜNG.....	14
4. NHẬN XÉT VÀ TỔNG KẾT	19
TÀI LIỆU THAM KHẢO.....	21

1. CÁC THUẬT TOÁN TÌM KIẾM

Việc cài đặt các thuật toán được thực hiện trên phần mềm Visual Studio Code, sử dụng ngôn ngữ lập trình Python 3. Sau đây là kết quả cài đặt của nhóm.

- a. Nhóm thuật toán tìm kiếm không có thông tin gồm: Thuật toán tìm kiếm theo chiều rộng (Breadth First Search) và chiều sâu (Depth First Search)

```
# BFS
def BFS(matrix, start, end):
    open_list, visited = [start], [start]
    closed_list = []
    while open_list:
        current = open_list.pop(0)
        closed_list.append(current)
        if current == end:
            return closed_list
        for neighbor in find_next(current, matrix):
            if neighbor not in visited:
                open_list.append(neighbor)
                visited.append(neighbor)
    return None
```

```
# DFS
def DFS(matrix, start, end):
    stack, visited = [start], [start]
    route = []
    while (stack):
        n = stack.pop()
        route.append(n)
        if n == end:
            return route
        for x in find_next(n, matrix):
            if x not in visited:
                visited.append(x)
                stack.append(x)
    return None
```

b. Nhóm thuật toán tìm kiếm có thông tin

- Các hàm hỗ trợ:

```
def heuristic_distance(node, end):
    return sqrt((node[0] - end[0])**2 + (node[1] - end[1])**2)

def heuristic_distance_bonus(node, end, bonus_points):
    for _, points in enumerate(bonus_points):
        if node == (points[0], points[1]):
            return points[2] + sqrt((node[0] - end[0])**2 + (node[1] - end[1])**2)
    return sqrt((node[0] - end[0])**2 + (node[1] - end[1])**2)

def actual_distance(node, start):
    return sqrt((node[0] - start[0])**2 + (node[1] - start[1])**2)
```

- Thuật toán tìm kiếm tham lam (Greedy Best First Search)

```
# Greedy
def Greedy(matrix, start, end, bonus_points):
    open_list = [start]
    closed_list = []
    while open_list:
        current = open_list.pop(0)
        closed_list.append(current)
        if current == end:
            return closed_list
        for neighbor in find_next(current, matrix):
            if (neighbor not in closed_list):
                open_list.append(neighbor)
        open_list.sort(key=lambda x: heuristic_distance_bonus(
            x, end, bonus_points))
    return None
```

(Thuật toán tìm kiếm tham lam có cải tiến khác biệt về vấn đề điểm thưởng, xem chi tiết ở [phần 3](#))

- Thuật toán tìm kiếm A*

```
# A*
def AStar(matrix, start, end):
    F, G, camefrom = {}, {}, {}
    G[start] = 0
    F[start] = heuristic_distance(start, end)
    opened, closed = [start], []

    while opened:
        current, currentF = None, None
        for pos in opened:
            if current is None or F[pos] < currentF:
                currentF = F[pos]
                current = pos
        if current == end:
            route = [current]
            while current in camefrom:
                current = camefrom[current]
            route.append(current)
            route.reverse()
            return route
        opened.remove(current)
        closed.append(current)

        for neighbor in find_next(current, matrix):
            if neighbor in closed:
                continue
            candidateG = G[current] + actual_distance(current, start)
            if neighbor not in opened:
                opened.append(neighbor)
            elif candidateG >= G[neighbor]:
                continue
            camefrom[neighbor] = current
            G[neighbor] = candidateG
            H = heuristic_distance(neighbor, end)
            F[neighbor] = G[neighbor] + H

    return None
```

- Một hàm có tính chất cực kỳ quan trọng, quyết định đến thành công của nhóm là hàm findNext, hàm chuyên biệt để lưu giữ toàn bộ các vị trí liền kề của vị trí hiện tại. Hàm này được thể hiện như sau:

```
def find_next(pos, matrix):  
    next = []  
    if (pos[0] - 1 >= 0 and matrix[pos[0]-1][pos[1]] != 'x'):  
        next.append((pos[0]-1, pos[1]))  
    if (pos[0] + 1 < len(matrix) and matrix[pos[0]+1][pos[1]] != 'x'):  
        next.append((pos[0]+1, pos[1]))  
    if (pos[1] + 1 < len(matrix[0]) and matrix[pos[0]][pos[1] + 1] != 'x'):  
        next.append((pos[0], pos[1] + 1))  
    if (pos[1] - 1 >= 0 and matrix[pos[0]][pos[1]-1] != 'x'):  
        next.append((pos[0], pos[1]-1))  
    return next
```

- Sau khi cài đặt xong các hàm này, nhóm tiến hành dựng bản đồ và cấu hình để chạy và hiển thị đường đi cho các bản đồ theo notebook được cung cấp bởi giảng viên hướng dẫn, chi tiết ở mục [Tài liệu tham khảo](#).

2. BẢN ĐỒ KHÔNG CÓ ĐIỂM THƯỞNG

Nhóm tiến hành dựng 5 bản đồ không có điểm thưởng để thực hiện so sánh kết quả và hiệu quả của các thuật toán tìm kiếm. Sau đây là bảng tóm tắt kết quả thực hiện tìm kiếm trên loạt bản đồ này:

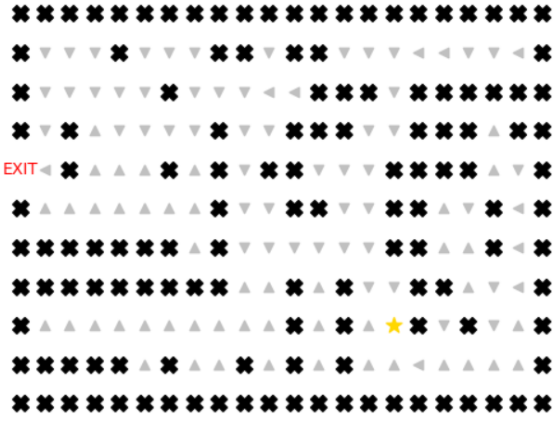
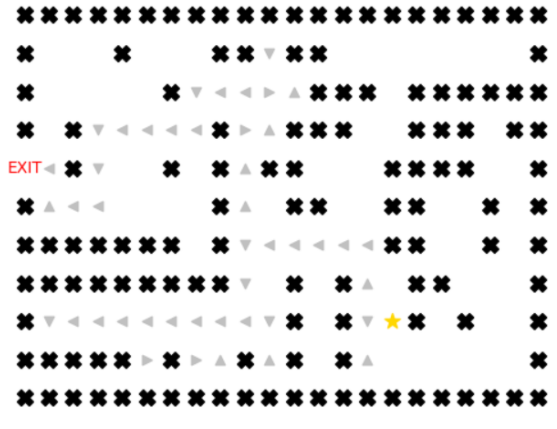
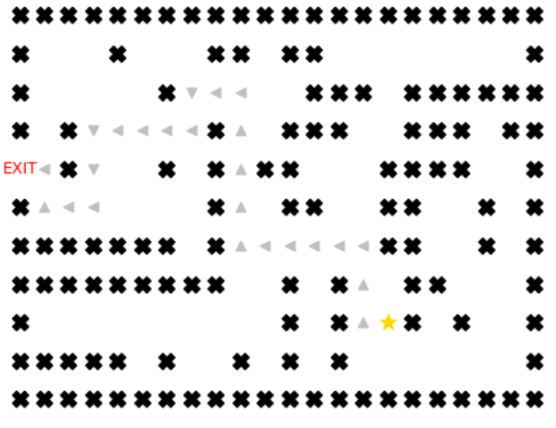
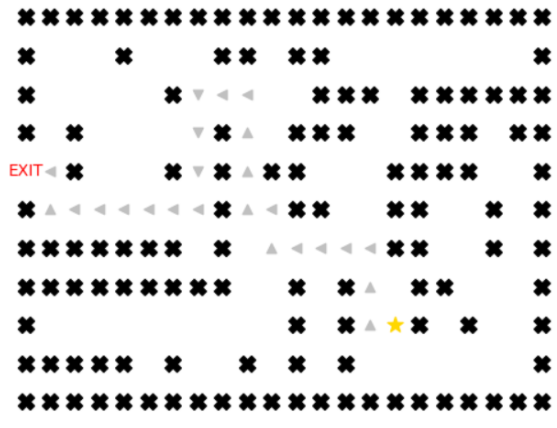
Bản đồ	Kích thước	Xuất phát	Lối thoát	Quãng đường di chuyển			
				BFS	DFS	Greedy	AStar
1	22 * 11	(8,15)	(4,0)	77	34	18	18
2	22 * 11	(8,15)	(4,0)	107	44	24	24
3	35 * 15	(9,32)	(4,0)	251	109	47	46
4	22 * 11	(1,20)	(8,0)	94	29	29	26
5	25 * 11	(1,13)	(6,0)	126	79	63	47

Tiếp theo là mô tả đường đi tương ứng với các bản đồ được lựa chọn.

- **Bản đồ số 1:** Vị trí xuất phát và vị trí lối thoát không quá đặc biệt.
 - Thuật toán tìm kiếm chiều rộng mang tính chất là đầy đủ - tìm thấy đường đi khi bản đồ ít ngõ ngách nhưng lại không tối ưu (trừ khi chi phí mỗi bước đi là 1 như hiện tại) → **Chi phí 77 là chấp nhận được**
 - Thuật toán tìm kiếm chiều sâu có đặc thù là không tối ưu (do luôn tìm về hướng trái nhất của bản đồ, bất kể chi phí) và cũng không hẳn là đầy đủ (trừ khi ít ngõ ngách hoặc ngõ ngách không quá sâu) → **Chi phí 34 là cao và không thực sự hiệu quả.**
 - Thuật toán tìm kiếm tham lam và tìm kiếm A* **cho kết quả tương đương**, khác nhau chính là lộ trình di chuyển (*khả năng phát hiện có đường tắt của A* tốt hơn, nhờ hàm heuristic có xét đến khoảng cách đến vị trí xuất phát*). Và sự khác nhau này sẽ là vấn đề khi sang bản đồ 2, vốn có vài sự điều chỉnh so với bản đồ 1.

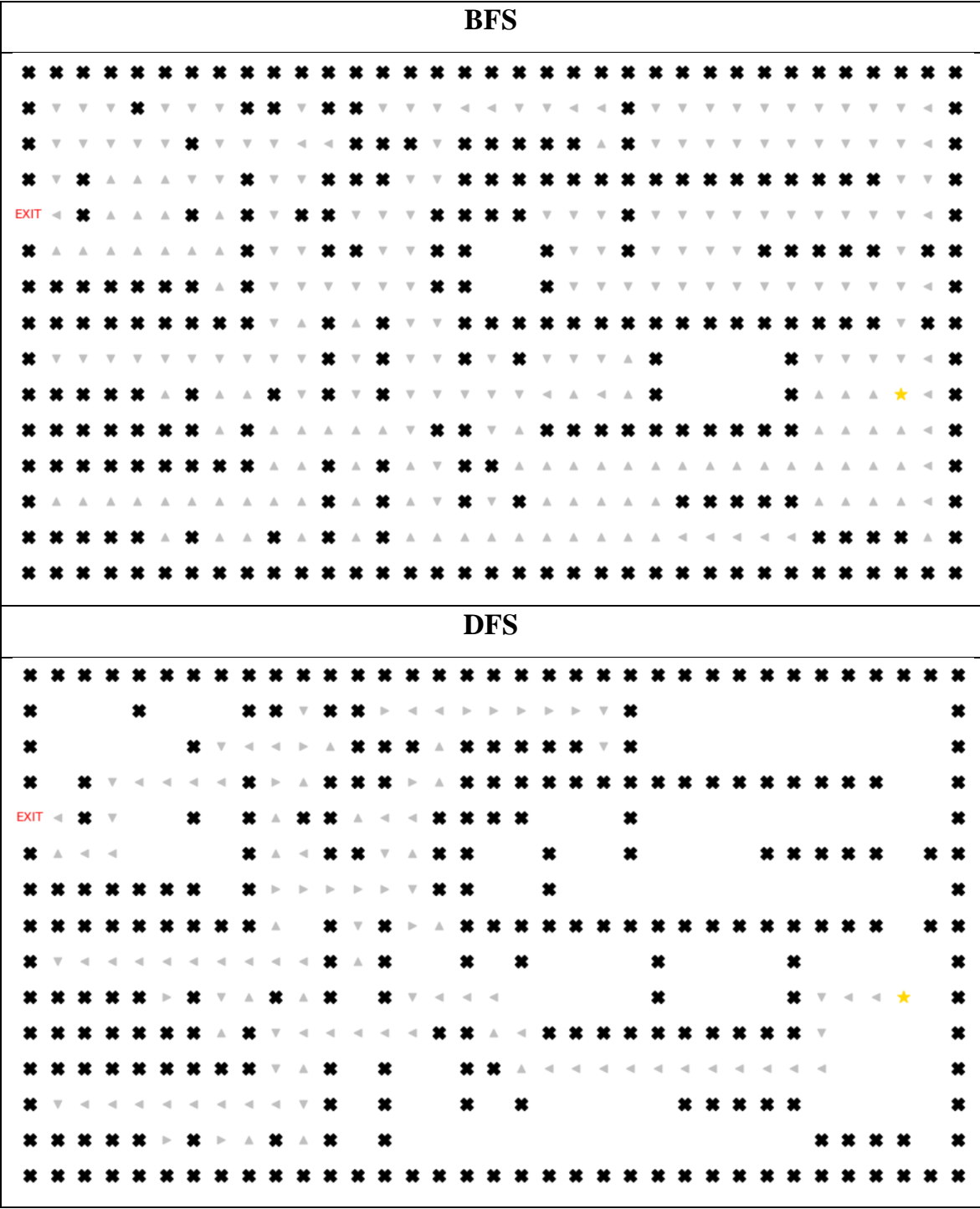
BFS	DFS
Greedy	A*

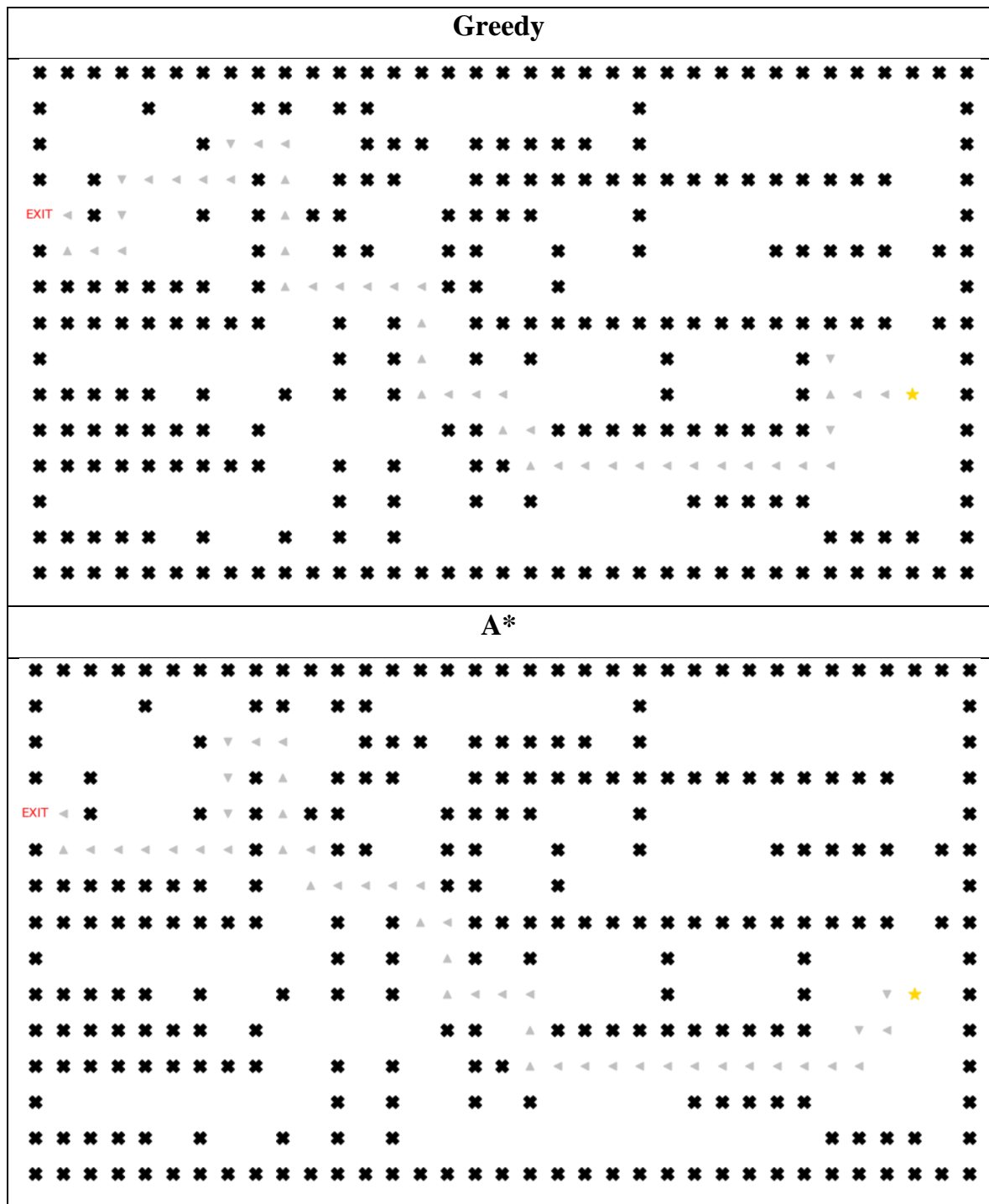
- **Bản đồ số 2:** Tương đương bản đồ 1. Khác biệt lớn nhất là chỉ còn con đường duy nhất - vượt qua khu vực tường chắn để đến được lối thoát. Lúc này phát sinh những vấn đề lớn:
 - **Thuật toán tìm kiếm chiều rộng** do phải quét toàn bộ các vị trí xung quanh điểm di chuyển, có chi phí bằng nhau để tìm lối thoát nhanh nhất → **Kết quả phải quét toàn bộ bản đồ (107), không hiệu quả.**
 - **Thuật toán tìm kiếm chiều sâu** cũng chịu ảnh hưởng, nhưng do xu hướng dò tìm về hướng trái nhất nên sự ảnh hưởng không quá nhiều → **Chi phí 44 là hợp lý.**
 - **Thuật toán tìm kiếm tham lam (Greedy) và A*** không chịu ảnh hưởng từ việc bị “mù đường” như BFS và DFS, tính toán lại lộ trình và cho ra kết quả khả quan. Dường như **Greedy thắng thế A*** khi lối thoát không lựa chọn đi vào ngõ ngách, dù chi phí của cả hai đều như nhau.

BFS	DFS
	
Greedy	A*
	

- **Bản đồ số 3:** Bản đồ kích cỡ lớn, vị trí xuất phát nằm sâu bên trong.

- Hai thuật toán tìm kiếm chiều rộng và chiều sâu đều thất bại với bản đồ có nhiều ngõ ngách này. Do tính chất của từng thuật toán nên kết quả **lộ trình quét gần hết bản đồ (251)** hay quét **gần một nửa các vị trí trống của bản đồ (109)** là không thực sự hiệu quả.
- So sánh giữa **thuật toán tìm kiếm tham lam (Greedy)** và **A*** thì lộ trình của A* tốt hơn, do việc lựa chọn hàm heuristic là **có thể chấp nhận được (admissable)** dẫn đến việc A* có tính chất đầy đủ và tối ưu. Heuristic được lựa chọn cũng đủ tốt nên Greedy chỉ khác biệt đôi chút về đoạn đầu và giữa của lộ trình, kết quả và tính chất tương đương A*.





- **Bản đồ số 4:** Vị trí xuất phát nằm gần góc bản đồ, vị trí lối thoát thay đổi.
 - o **Thuật toán tìm kiếm theo chiều rộng (BFS)** tiếp tục quét các vị trí xung quanh mà không đào sâu về hướng nào, **chiều dài lộ trình 94 vẫn không phải là tốt nhất**, nhưng so với các bản đồ trước, do ít ngõ ngách hơn nên thuật toán cũng hiệu quả hơn một chút.

- Điều khiến nhóm khá bất ngờ chính là việc **thuật toán tìm kiếm chiều sâu (DFS)** có thể đạt được kết quả tương đương **thuật toán tìm kiếm tham lam (Greedy)**, cả về mặt lộ trình và thậm chí cả về đường đi. Chiều dài lộ trình 29 rất hiệu quả với cả DFS (do tối ưu hướng trái nhất của bản đồ) lẫn Greedy (đường đi dựa trên Heuristic)
- **Thuật toán tìm kiếm A*** vẫn phát huy tác dụng của mình là thuật toán có thể tận dụng tốt hàm heuristic được tính toán bởi **tổng khoảng cách ước lượng và khoảng cách thực tế**, tiết kiệm các bước đi thừa vào sâu trong các nhánh không dẫn đến lối thoát. Kết quả 26 là kết quả tốt nhất trong các con số tại bản đồ này.

BFS	DFS
<pre> ***** * *v*v* *v*v*v*v*v* *v*v*v*v*v* * v*v*v*v*v*v*v*v*v*v*v*v*v*v*v*v * *v*v*v*v*v*v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v*v*v ***** EXIT * * * *v*v*v*v*v*v*v*v*v*v*v*v ***** </pre>	<pre> ***** * * *v*v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v ***** EXIT * * * *v*v*v*v*v*v*v*v*v*v*v*v ***** </pre>
Greedy	A*
<pre> ***** * * *v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v ***** EXIT * * * *v*v*v*v*v*v*v*v*v*v*v*v ***** </pre>	<pre> ***** * * *v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * * *v*v*v*v*v*v*v*v*v*v*v*v * * *v*v*v*v*v*v*v*v*v*v*v ***** EXIT * * * *v*v*v*v*v*v*v*v*v*v*v*v ***** </pre>

- **Bản đồ số 5:** Bản đồ có nhiều khu vực tường chắn, vị trí xuất phát nằm sâu về bên trong và chỉ có 1 hướng xuất phát duy nhất. Đây là nơi chứng kiến những sự khác biệt nhất giữa các thuật toán

- **Thuật toán tìm kiếm chiều rộng (BFS)** thất bại do tính không tối ưu của nó nên vẫn quét xung quanh các vị trí lân cận để rà đường đi.
- **Thuật toán tìm kiếm chiều sâu (DFS)** không tối ưu (do ngược hướng di chuyển chủ đạo) cũng không đầy đủ (do có nhiều ngách sâu không dẫn đến lối thoát) nên đưa ra lộ trình “không bình thường”.
- **Thuật toán tìm kiếm tham lam (Greedy)** chịu ảnh hưởng từ hàm heuristic, tuy nhiên cũng giống như DFS, vì chính sự tham lam của nó (phụ thuộc hoàn toàn vào heuristic mà không ước tính vị trí thực) nên Greedy cũng chịu cảnh đi vào ngách sâu không phải lối thoát trước khi tìm thấy lộ trình chính xác.
- **Thuật toán tìm kiếm A*** nhờ hàm heuristic đầy đủ hơn, khắc phục được những bất cập của các thuật toán khác và cho ra lộ trình tốt nhất.

BFS	DFS
Greedy	A*

3. BẢN ĐỒ CÓ ĐIỂM THƯỞNG

Đối với chiến lược tìm kiếm đường đi trong bản đồ có điểm thưởng, sau khi xem xét các hàm heuristic thì nhóm nhận thấy việc sử dụng hàm khoảng cách Euclid phát huy tác dụng tốt nhất trong quá trình tìm kiếm đường đi (có thông tin) cho các mê cung như đã thể hiện ở [phần 2](#).

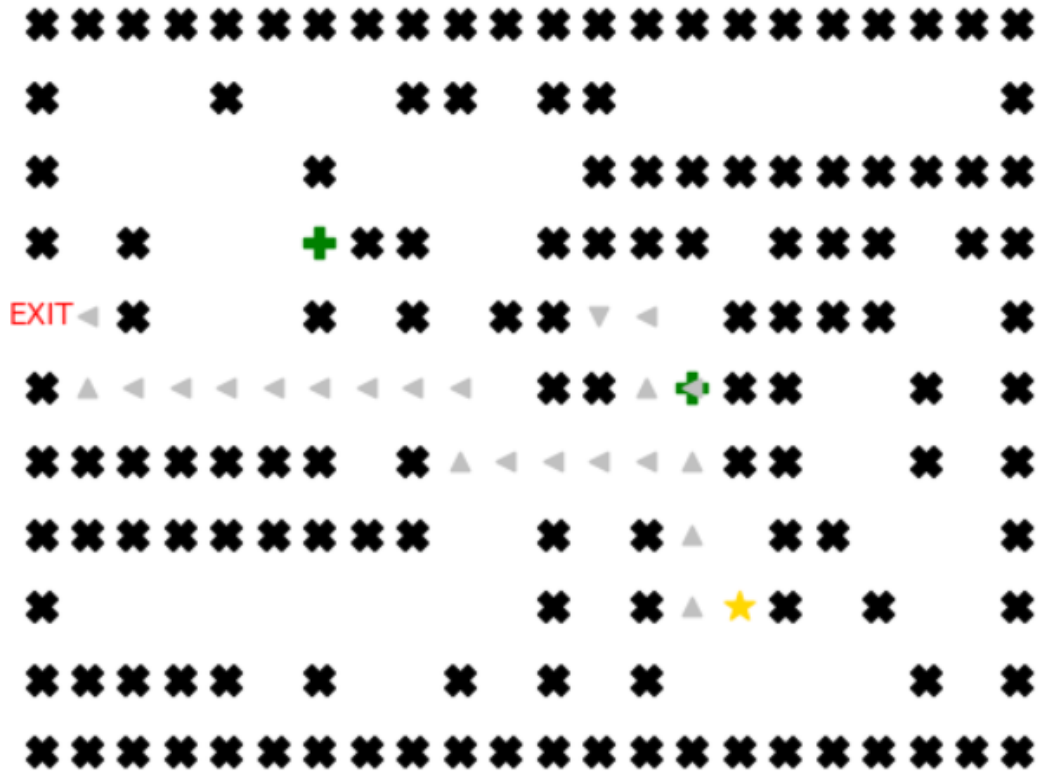
Với tốc độ xử lý nhanh và độ chính xác cao, việc sử dụng hàm khoảng cách Euclid phát huy tác dụng cực kỳ tốt. Sau khi khảo sát ba bản đồ, nhóm quyết định sử dụng thuật toán tìm kiếm tham lam (Greedy Best-First Search) mở rộng cho việc di chuyển vào các khu vực có điểm thưởng. Mục tiêu lớn nhất của nhóm chính là có thể tối ưu thuật toán này, tạo ra hành trình có chi phí không chênh lệch quá lớn (hoặc thậm chí là tương đương) so với một thuật toán tìm kiếm có thông tin khác là thuật toán tìm kiếm A* (với hàm heuristic không có điểm thưởng).

Sau đây, nhóm sẽ thể hiện kết quả thông qua bảng này, đồng thời đưa ra nhận xét đối với từng bản đồ.

Bản đồ	Kích thước	Xuất phát	Lối thoát	Bản đồ so sánh	Điểm thưởng			Quãng đường	
					Đã qua	Vị trí	Giá trị	Greedy (Bonus)	A*
6	22 * 11	(8,15)	(4,0)	1	1/2	(5,14)	1	21	18
7	25 * 11	(1,13)	(6,0)	5	3/5	(5,4), (5,10), (4,19)	7	61	42*
8	35 * 15	(9,32)	(4,0)	3	1/10	(8,29)	2	45	46

Sau đây là phần nhận xét trên các bản đồ.

- **Bản đồ số 6:** Bản đồ này tương đương với bản đồ số 1, với hai điểm thưởng: **(3,6)** với trọng số **3** và **(5,14)** với trọng số **1**. Ngay khi áp dụng hàm heuristic được cải tiến (**heuristic_distance_bonus**, [phần 2](#)), hàm này tiến hành quét các điểm thưởng. Tuy nhiên, kết quả không tốt hơn khi sử dụng Greedy với hàm heuristic thường, **thậm chí còn hao chi phí hơn**. Mặc dù vậy, nếu so với thuật toán tìm kiếm A*, **chi phí 21 là có thể chấp nhận được**.



- **Bản đồ số 7:** Bản đồ này tương đương với bản đồ số 5, với năm điểm thưởng ở *bảng 1* bên dưới. Điều bất ngờ chính là, dù cho Greedy với hàm heuristic cải tiến **ăn được 3 điểm thưởng**, nhưng **kết quả vẫn không hiệu quả (61)**. So với thuật toán tìm kiếm A^* , *vốn không được tính chính trong phân tích toán điểm thưởng*, A^* **vẫn ăn trực tiếp được hai điểm thưởng (*)**, góp phần giảm chi phí di chuyển tối đa.

Greedy (với điểm thưởng)	A^*
<pre> xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx x x xx xx * >>>>>>>>> x x + x >>>> xxxxxxxxxxxx >>>> x x >>>>>>>> x xxx xxx >>> x x >> x >> xx xxxxxx + >> x x >>>> + >> + xx xxx >> x EXIT- xxxxxx x >>>>>>>> x >> x xxxxxxxxxxx > x > xx >> x x >>>>>>>>>>>> x > xx >> x xxxxxxxx > >> x x >>>>>>>> + x xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx </pre>	<pre> xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx x x xx xx * >>>>>>>>> x x + x >>>> xxxxxxxxxxxx >>>> x x x >>>> x >> xxx xxx >> x x x >>>> x >> xxxxxx + >> x x >>>> + >>>>>>>> + xx >> x EXIT- xxxxxx x >>>>>>>> x >> x xxxxxxxxxxx x >> >>>> >> >> x x >>>>>>>>>>>> x > xx >> x xxxxxxxx x x x x >>>>>>>> + x xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx </pre>

- **Bản đồ số 8:** Bản đồ lớn, 10 điểm thưởng như **bảng 2** bên dưới, tương đương bản đồ số 3. Điều thú vị đã đến: Dù chỉ ăn duy nhất 1 điểm thưởng, nhưng với trọng số tại điểm thưởng là 2, Greedy đã giảm chi phí xuống **thấp hơn** so với thuật toán A*, mặc dù mức giảm này là không đáng kể.

Greedy (với điểm thưởng)																																					
A*																																					

Bảng 1: Vị trí và trọng số của 5 điểm thưởng

STT	Vị trí	Số điểm
1	(2,2)	2
2	(5,4)	2
3	(5,10)	3
4	(4,19)	2
5	(9,21)	1

Bảng 2: Vị trí và trọng số của 10 điểm thưởng

STT	Vị trí	Số điểm
1	(2,2)	2
2	(3,3)	1
3	(8,6)	1
4	(6,7)	3
5	(9,10)	1
6	(9,19)	2
7	(6,24)	1
8	(1,29)	2
9	(8,29)	2
10	(13,23)	3

4. NHẬN XÉT VÀ TỔNG KẾT

Thông qua phần báo cáo, nhóm rút ra một số điểm sau:

- **Đối với các thuật toán tìm kiếm không có thông tin:**
 - **Thuật toán tìm kiếm theo chiều rộng (BFS)** thích hợp với các khu vực địa hình có ít ngõ ngách, ít chướng ngại vật, phạm vi nhỏ (**bản đồ 1, 4**).
 - **Thuật toán tìm kiếm theo chiều sâu (DFS)** thích hợp với các khu vực địa hình có số lượng ngõ ngách giới hạn, các ngõ ngách có độ sâu tương đối, ít lối tắt, phạm vi trung bình (**bản đồ 2,4,5**).
- **Đối với các thuật toán tìm kiếm có thông tin:**
 - **Thuật toán tìm kiếm tham lam (Greedy)** và tham lam có cải tiến thích hợp với các khu vực địa hình có số lượng ngõ ngách tương đối lớn, các ngõ ngách có độ sâu lớn, nhiều chướng ngại vật, nhiều lối tắt. Tuy nhiên, Greedy sẽ không phù hợp với các dạng địa hình có *quá nhiều ngõ cụt kết hợp với ngõ sâu (bản đồ 8)*.
 - **Thuật toán tìm kiếm A*** thích hợp với hầu hết các loại địa hình, tuy nhiên hạn chế cần chú ý về lộ trình di chuyển là A* thận trọng đánh giá dựa trên một số tiêu chí cụ thể (**ở đây là hàm heuristic**), do đó *nếu đánh giá sai, A* có khả năng sẽ gặp hạn chế như Greedy*, khi đi vào nhiều ngõ cụt không cần thiết mà không đến được đích (**bản đồ 3, 5,6,7**).
- **Đối với hàm heuristic**, do thời gian nghiên cứu bị giới hạn nên nhóm chọn nâng cấp hàm thuật toán tìm kiếm tham lam để dùng cho các bản đồ có điểm thưởng. Tuy vậy, nhóm có thể tóm tắt việc nghiên cứu hàm heuristic bằng một số điều sau:
 - **Thứ nhất**, hàm heuristic cần dùng là heuristic chỉ có thể di chuyển theo 4 hướng lên, xuống, trái, phải. Như vậy nhóm sử dụng hai hàm heuristic là heuristic Manhattan và heuristic Euclid.

- **Thứ hai**, đối với heuristic Manhattan, do đặc thù là tính toán dựa trên khoảng cách tuyệt đối, do đó trong quá trình khảo sát đã không đạt được hiệu quả tối ưu nhất và dẫn đến sai sót.
- **Thứ ba**, đối với heuristic Euclid, mặc dù ban đầu nhóm không dự định sử dụng hàm này (do một số lí do mà chủ yếu là hàm **sqrt**, vốn không thể chạy hiệu quả trong một số tình huống như địa hình có nhiều chướng ngại), nhưng do heuristic Manhattan không tối ưu, do đó nhóm quyết định khảo sát hàm này.
- **Kết quả**, đối với các địa hình có chi phí như nhau cho mỗi bước đi, heuristic Euclid phát huy tác dụng tốt hơn trong quá trình tính toán đường đi, độ chính xác cũng vì thế tăng lên. Có thể kết luận dựa trên tám bản đồ địa hình được cung cấp, mức độ tối ưu đạt từ 75% trở lên.

TÀI LIỆU THAM KHẢO

1. Slides bài giảng [“Bài toán tìm kiếm”](#), Nguyễn Ngọc Đức, Trường Đại học Khoa học Tự nhiên – Đại học Quốc gia Thành phố Hồ Chí Minh
2. Slides bài giảng [“Tìm kiếm có định hướng”](#), Nguyễn Ngọc Đức, Trường Đại học Khoa học Tự nhiên – Đại học Quốc gia Thành phố Hồ Chí Minh
3. Các đoạn mã nguồn đọc file để vẽ bản đồ và mô hình hóa bằng Matplotlib được tham khảo từ notebook tại [đây](#).
4. [Artificial Intelligence – Search](#)
5. [Artificial Intelligence – Informed Search](#)