TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN MÔN: TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI

Tìm hiểu các thuật toán tìm kiếm mù và ứng dụng nó vào bài toán người lái đò

Giáo viên hướng dẫn:	Trần Thanh Huân
Lớp:	20242IT6094002
Nhóm:	2
Thành viên nhóm:	Nguyễn Thiện Vinh-2023604053
	Nguyễn Huyền Trang-2023603130
	Nguyễn Tiến Anh Nhật-2022602497
	Nguyễn Ngọc Sơn-2023604954

Hà Nội, 2025

Lời mở đầu

Trong lĩnh vực Trí tuệ nhân tạo và Tin học, các thuật toán tìm kiếm đóng vai trò then chốt trong việc giải quyết những bài toán phức tạp, đặc biệt là những bài toán đòi hỏi tìm kiếm lời giải tối ưu trong một không gian trạng thái rộng lớn. Trong số đó, nhóm các thuật toán tìm kiếm mù (Blind Search) là những phương pháp cơ bản nhưng rất hiệu quả, được sử dụng khi không có hoặc rất ít thông tin bổ sung về đích đến hoặc cấu trúc của không gian trạng thái. Các thuật toán này trở thành công cụ hữu ích trong việc tiếp cận các bài toán mà giải pháp không thể suy luận trực tiếp mà cần phải khám phá toàn bộ hoặc một phần không gian trạng thái.

Bài toán "Người lái đò" là một ví dụ điển hình và kinh điển trong Trí tuệ nhân tạo, thường được sử dụng để minh họa cho việc biểu diễn trạng thái, xây dựng không gian tìm kiếm và áp dụng các chiến lược tìm kiếm để tìm ra lộ trình hợp lý. Bài toán đặt ra yêu cầu người lái đò phải đưa một nhóm đối tượng qua sông mà không vi phạm các điều kiện ràng buộc, từ đó tạo thành một mô hình lý tưởng để thực hành triển khai các thuật toán tìm kiếm mù như Breadth-First Search (BFS), Depth-First Search (DFS), và Uniform-Cost Search (UCS).

Việc nghiên cứu và ứng dụng các thuật toán tìm kiếm mù vào bài toán người lái đò không chỉ giúp chúng em hiểu rõ hơn về cơ chế hoạt động và ưu nhược điểm của từng chiến lược tìm kiếm, mà còn rèn luyện kỹ năng phân tích, mô hình hóa bài toán và tư duy giải quyết vấn đề theo hướng logic, hệ thống.

Với những lý do trên, nhóm chúng em đã lựa chọn đề tài này làm nội dung nghiên cứu cho bài tiểu luận môn *Trí tuệ nhân tạo* dưới sự hướng dẫn của thầy Trần Thanh Huân. Trong quá trình thực hiện, chúng em đã cố gắng tìm hiểu, thảo luận và phối hợp làm việc nghiêm túc để hoàn thành bài tiểu luận đúng theo yêu cầu và đạt chất lượng tốt nhất có thể.

Chúng em xin gửi lời cảm ơn chân thành đến thầy Trần Thanh Huân, người đã tận tình giảng dạy, hướng dẫn và tạo điều kiện thuận lợi cho chúng em trong suốt quá trình học tập và thực hiện đề tài này. Hy vọng rằng bài tiểu luận sẽ góp phần làm rõ hơn giá trị ứng dụng của các thuật toán tìm kiếm mù trong việc giải quyết những bài toán cổ điển lẫn thực tiễn trong lĩnh vực Trí tuệ nhân tạo.

Chúng em xin chân thành cảm ơn!

DANH MỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG 1. KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM MÙ

Chương này giới thiệu cách biểu diễn bài toán trong không gian trạng thái, từ không gian này chuyển sang biểu diễn bằng đồ thị; các chiến lược tìm kiếm mù và sau cùng là các kỹ thuật tìm kiếm theo kinh nghiệm được áp dụng cho bài toán có độ phức tạp tính toán cấp hàm mũ [2].

1.1. Phân tích vấn đề

Khi giải quyết bài toán phương pháp tìm kiếm, trước hết ta phải xác định được **không gian tìm kiếm** (bao gồm tất cả các đối tượng mà trên đó thực hiện việc tìm kiếm). Nó có thể là không gian liên tục và nó cũng có thể là không gian các đối tượng rời rạc. Như vậy, ta sẽ xét việc biểu diễn một bài toán trong không gian trạng thái sao cho việc giải quyết bài toán này được quy về việc tìm kiếm lời giải trong không gian trạng thái. Ta có thể sử dụng các khái niệm **trạng thái** và **toán tử** để biểu diễn bài toán đang xét.

Phương pháp giải quyết vấn đề dựa trên khái niệm trạng thái và toán tử được gọi là cách tiếp cận giải quyết vấn đề nhờ không gian trạng thái.

1.2. Không gian trạng thái

1.2.1. Mô tả trạng thái

Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phù hợp với bản chất vật lý của bài toán (có thể sử dụng các xâu ký hiệu, véctơ, mảng hai chiều, cây, danh sách, ...) [3].

Mỗi **trạng thái** chính là mỗi **hình trạng** của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối.

Ví dụ 2.1: Bài toán trò chơi 8 số

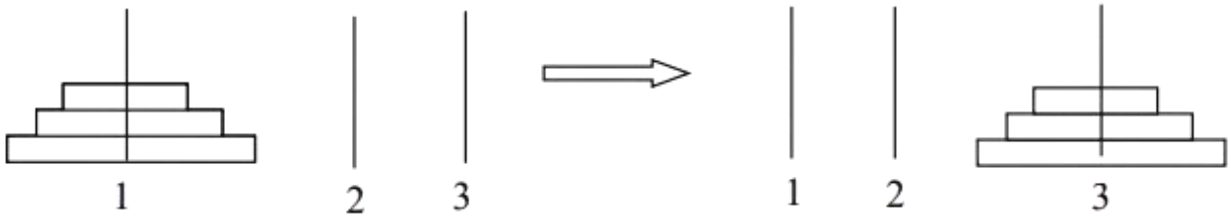
Trong bảng ô vuông 3 hàng, 3 cột, mỗi ô chứa một số nằm trong phạm vi từ 1 đến 8 sao cho không có 2 ô có cùng giá trị, có một ô trong bảng bị trống (không chứa giá trị nào cả). Xuất phát từ một sắp xếp nào đó các số trong bảng, hãy dịch chuyển ô trống sang phải, sang trái, lên trên hoặc xuống dưới (nếu có thể được) để đưa bảng ban đầu về bảng quy ước trước.

Trạng thái đầu Trạng thái đích

Hình 1.1. Bài toán 8 số

Mỗi hình trạng trong bài toán này là một cách sắp xếp các con số. Ta có thể dùng mảng hai chiều kích thước 3x3 hoặc mảng một chiều kích thước 9 để biểu diễn cho mỗi trạng thái trong máy tính.

Ví dụ 2.2: Bài toán tháp Hà Nội



Hình 1.2: Bài toán tháp Hà Nội với n = 3

Cho 3 cọc 1, 2, 3. Ở cọc 1 ban đầu có n đĩa sắp xếp theo thứ tự từ nhỏ đến lớn. Hãy dịch chuyển n đĩa đó sang cọc 3 sao cho:

- Mỗi lần chuyển một đĩa.
- Trong mỗi cọc không được đặt đĩa to ở trên đĩa nhỏ hơn trong bất cứ tình huống nào.

Trong bài toán trên, mỗi trạng thái là một bộ (ijk) với ý nghĩa: Đĩa C (đĩa lớn nhất) ở cọc i, đĩa B ở cọc j, đĩa A (đĩa bé nhất) ở cọc k. Trạng thái đầu là (111) còn trạng thái đích là (333).

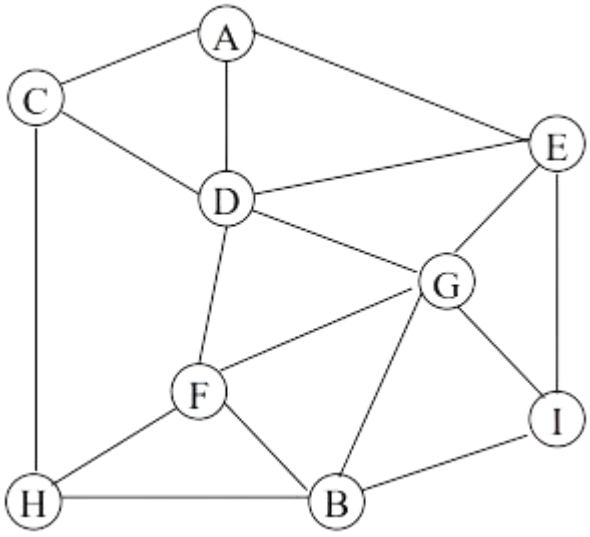
Với bài toán này, ta cũng có thể dùng mảng hai chiều để biểu diễn: 3 cột tương ứng với 3 cọc, số dòng tương ứng với số đĩa. Cụ thể, trong trường hợp này ta dùng mảng 3x3.

Trạng thái đầu Trạng thái đích

Hình 1.3. Trạng thái trong bài toán tháp Hà Nội

Ví dụ 2.3: Bài toán khách du lịch

Một khách du lịch có trong tay bản đồ mạng lưới giao thông nối các thành phố trong một vùng lãnh thổ (hình 1.4). Du khách đang ở thành phố A và anh ta muốn tìm đường đi tới thành phố B.



Hình 1.4. Tìm đường đi từ A đến B

Trong bài toán này, các thành phố có trong bản đồ là các trạng thái. Thành phố A là trạng thái đầu còn thành phố B là trạng thái kết thúc. Với bài toán này, ta có thể sử dụng cách biểu diễn của đồ thị (ma trận kề, ma trận trọng số, danh sách cạnh, danh sách kề).

1.2.2. Toán tử

Toán tử là các phép biến đổi từ trạng thái này sang trạng thái khác.

Có hai cách dùng để biểu diễn các toán tử:

- Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.

- Biểu diễn dưới dạng các luật sản xuất S → A, có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

Trong **ví dụ 2.1**, mỗi toán tử là cách chuyển ô trống, có 4 kiểu toán tử: chuyển ô trống lên trên, xuống dưới, sang trái, sang phải. Tuy nhiên, đối với một số trạng thái nào đó, một toán tử nào đó có thể không áp dụng được, chẳng hạn, nếu ô trống nằm ở cột đầu tiên thì ô trống không thể sang trái được.

Trong **ví dụ 2.2**, toán tử là cách chuyển đĩa từ cọc này sang cọc khác, mỗi lần chuyển một đĩa và không được đặt đĩa to ở trên đĩa nhỏ

Chẳng hạn như:

(ijk) → (ijj) (Chuyển đĩa A từ cọc k sang cọc j)

(ijk) → (iik) (Chuyển đĩa B từ cọc j sang cọc i)

(1 1 1) → (1 1 2)

(1 1 1) → (1 1 3)

Trong **ví dụ 2.3**, mỗi toán tử là hành động đi từ thành phố này tới các thành phố khác.

1.2.3. Không gian trạng thái của bài toán tìm kiếm

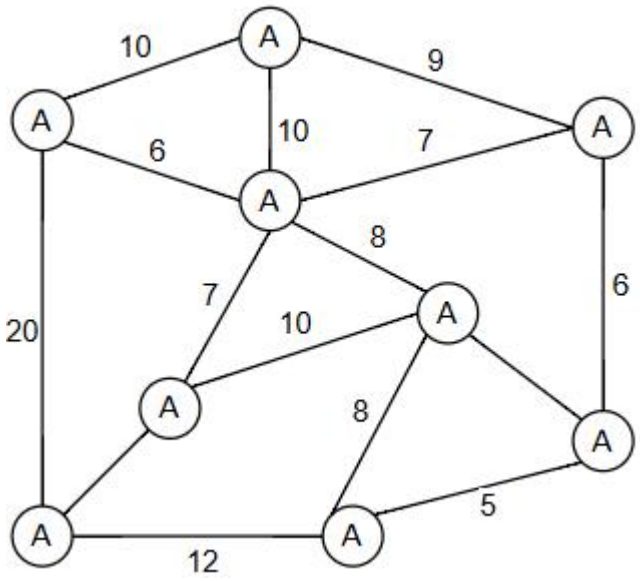
Từ các ví dụ 2.1, 2.2, 2.3 ta có thể thấy nhiều bài toán đều có dạng "tìm đường đi trong đồ thị", hay nói một cách hình thức hơn là "*xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến một đỉnh nào đó*". Từ đây, ta có bài toán phát biểu trong không gian trạng thái.

Bài toán S: Cho trước hai trạng thái T_0 và T_G, hãy xây dựng chuỗi trạng thái T_0, T_1, T_2, …, T_{n-1}, $T_n = T_G$ sao cho:

- $\sum_{i=1}^{n} \cos t(T_{i-1}, T_i)$ thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).

Trong đó, T_i thuộc tập hợp S (gọi là không gian trạng thái – state space) bao gồm tất cả các trạng thái có thể có của bài toán, và cost(T_{i-1}, T_i) là **chi phí** để **biến đổi** từ trạng thái T_{i-1} sang trạng thái T_i. Khi nói đến một biến đổi cụ thể từ T_{i-1} sang T_i ta sẽ dùng thuật ngữ **hướng đi** (với ngụ ý nói về sự lựa chọn).

Mô hình chung của các bài toán phải giải quyết bằng phương pháp tìm kiếm lời giải. Không gian tìm kiếm là một tập hợp trạng thái -–tập các nút của đồ thị. Chi phí cần thiết để chuyển từ trạng thái T_{i-1} này sang trạng thái T_i được biểu diễn dưới dạng các con số nằm trên cung nối giữa hai nút.



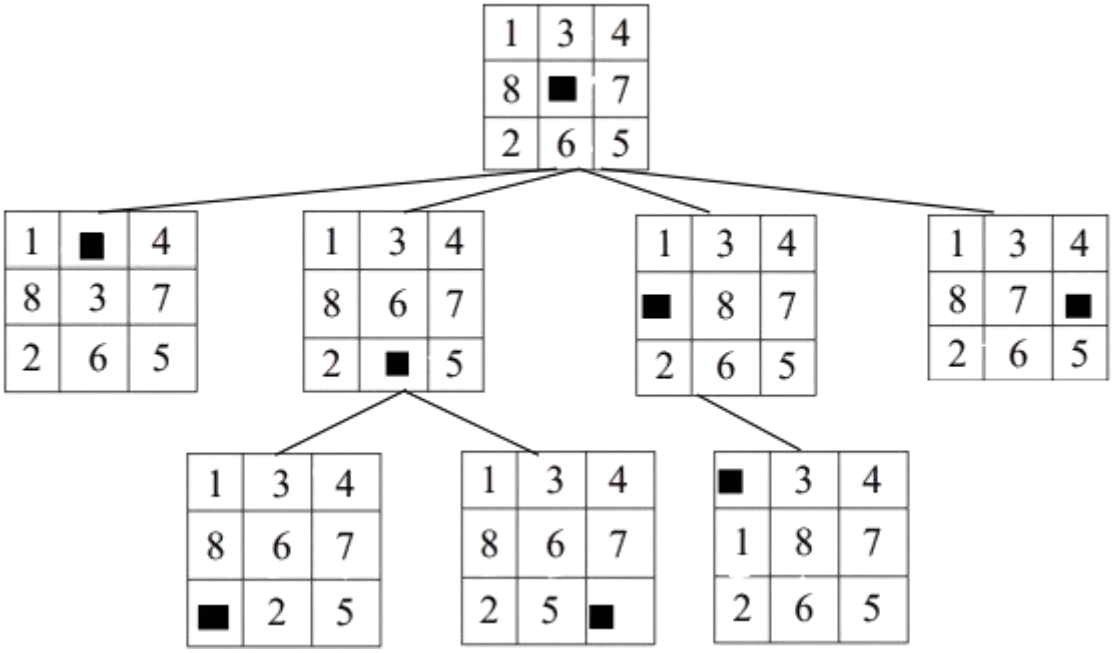
Hình 1.5. Đồ thị có trọng số

Không gian trạng thái	**Đồ thị**
- Trạng thái đầu	- Đỉnh đầu
- Trạng thái đích	- Đỉnh đích
- Toán tử dịch chuyển	- Cung
- Dãy các trạng thái	- Đường đi
- Bài toán S	- Bài toán G

Hình 1.6. Bảng không gian trạng thái

Chẳng hạn, với đồ thị trong hình 2.5, ta có thể phát biểu bài toán. Hãy tìm đường đi ngắn nhất từ đỉnh A đến đỉnh B.

Đa số các bài toán thuộc dạng mà chúng ta đang mô tả đều có thể được biểu diễn dưới dạng đồ thị. Trong đó, mỗi trạng thái là một đỉnh của đồ thị. Tập hợp S bao gồm tất cả các trạng thái chính là tập hợp bao gồm tất cả các đỉnh của đồ thị. Việc biến đổi từ trạng thái T_{i-1} sang trạng thái T_i là việc đi từ đỉnh đại diện cho T_{i-1} sang đỉnh đại diện cho T_{i-1} sang đỉnh đại diện cho T_i theo cung nối giữa hai đỉnh này.



Hình 1.7: Một phần đồ thị biểu diễn trò chơi 8 số

Bài toán G: Cho đỉnh đầu S_0 và tập các đỉnh Goal. Hãy tìm đường đi p (tối ưu) nào đó từ đỉnh S_0 đến đỉnh nào đó thuộc tập Goal.

Từ những phân tích ở trên, ta có thể thấy được sự tương đương giữa không gian trạng thái và đồ thị được tổng hợp trong Bảng 1.1.

1.3. CÁC THUẬT TOÁN TÌM KIẾM MÙ

Trong các kỹ thuật tìm kiếm này, không có một sự hướng dẫn nào cho sự tìm kiếm, mà ta chỉ phát triển các trạng thái một cách hệ thống từ trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có ba kỹ thuật tìm kiếm mù cơ bản, đó là tìm kiếm theo chiều rộng (Breadth First Search), tìm kiếm theo chiều sâu (Depth First Search) và tìm kiếm sâu dần [4].

Nhận xét: Trong tìm kiếm mù, ta chọn trạng thái để phát triển theo thứ tự mà chúng được sinh ra.

1.3.1. Thuật toán tìm kiếm theo chiều sâu (Depth First Search)

a. Tư tưởng của chiến lược tìm kiếm theo chiều sâu

- Từ đỉnh xuất phát duyệt một đỉnh kề.
- Các đỉnh của đồ thị được duyệt theo các nhánh đến nút lá.
- Nếu chưa tìm thấy đỉnh T_G thì quay lui tới một đỉnh nào đó để sang nhánh khác.

- Việc tìm kiếm kết thúc khi tìm thấy đỉnh T_G hoặc đã hết các đỉnh.

b. Thuật toán tìm kiếm theo chiều sâu

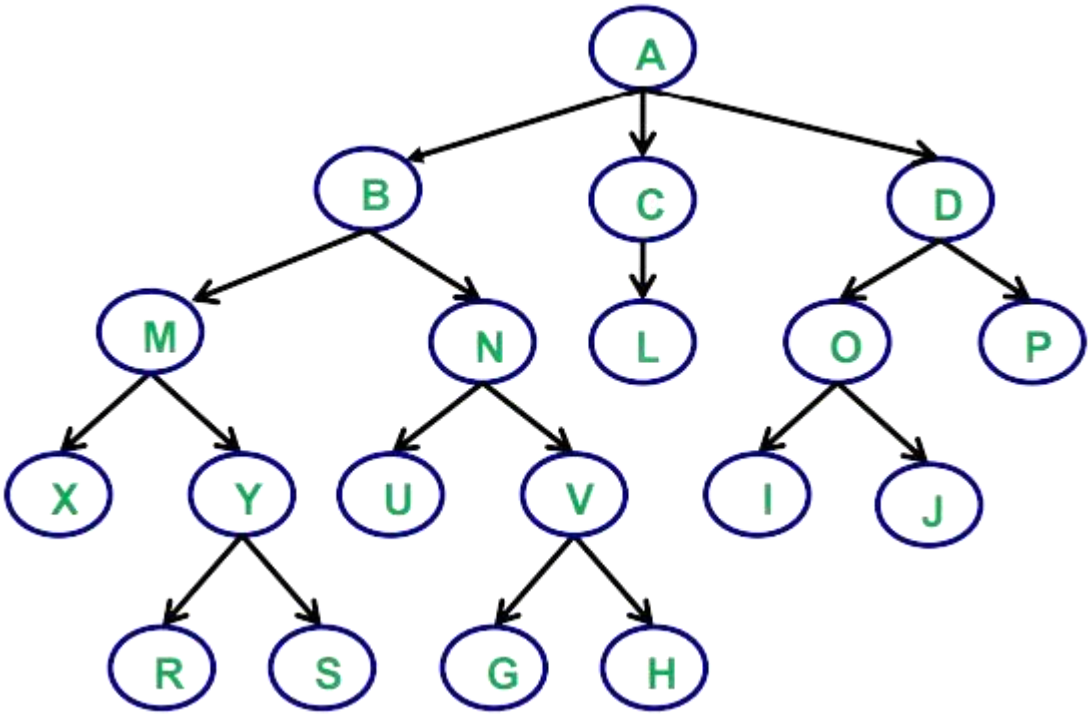
Khi lưu trữ thì sử dụng hai danh sách DONG và MO trong đó:

- DONG: Chứa các đỉnh đã xét, hoạt động theo kiểu FIFO *(hàng đợi)*.
- MO: Chứa các đỉnh đang xét, hoạt động theo kiểu LIFO *(ngăn xếp)*.

```
1. MO =∅;            MO = MO ∪ {T0}
2.      while (MO != ∅)
        {
                n = get(MO) //lấy đỉnh đầu trong danh sách MO
                if (n == TG) //nếu n là trạng thái kết thúc
                        return TRUE //tìm kiếm thành công, dừng
                DONG = DONG ∪ {n} //đánh dấu n đã được xét
                for các đỉnh kề v của n
                        if (v chưa được xét) //v chưa ở trong DONG
                        MO = MO ∪ {v} //đưa v vào đầu danh sách MO
                                father(v) = n //lưu lại vết đường đi từ n đến v
        }
```

c. Ví dụ thuật toán tìm kiếm theo chiều sâu

Cho đồ thị như hình vẽ sau:



Hình 1.8. Đồ thị tìm kiếm

Đỉnh đầu T_O = A, T_G = {R}

Tìm đường đi p từ T_O đến T_G bằng phương pháp tìm kiếm theo chiều sâu?

n	B(n)	MO	DONG
		A	
A	B, C, D	B, C, D	A
B	M, N	M, N, C, D	A, B
M	X, Y	X, Y, N, C, D	A, B, M
X	∅	Y, N, C, D	A, B, M, X
Y	R, S	R, S, N, C, D	A, B, M, X, Y
R	→ là đích → là dừng		

Hình 1.9. Bảng duyệt đồ thị DFS

Xây dựng đường đi có hành trình: p = A → B → M → Y → R

Nhận xét:

- Nếu trong đồ thị G tồn tại đường đi từ T_O đến 1 đỉnh $T_G \in$ Goal thì hàm DFS sẽ dừng lại và cho đường đi p có độ dài có thể không ngắn nhất.

- Với DFS các đỉnh được duyệt theo từng nhánh (theo chiều sâu).

- Thuật toán DFS có độ phúc tạp O(b^d) với b là bậc của cây và d là chiều sâu của cây. Tuy nhiên trong trường hợp xấu nhất cũng là O(b^d).

1.3.2. Thuật toán tìm kiếm theo chiều rộng (Breadth First Search)

a. Tư tưởng của chiến lược tìm kiếm theo chiều rộng

- Từ đỉnh xuất phát duyệt tất cả các đỉnh kề.
- Làm tương tự với các đỉnh vừa được duyệt.
- Quá trình duyệt kết thúc khi tìm thấy đỉnh T_G hoặc đã hết các đỉnh để duyệt.

b. Thuật toán tìm kiếm theo chiều rộng

Khi lưu trữ thì sử dụng hai danh sách DONG và MO hoạt động theo kiểu FIFO *(hàng đợi)*.

- DONG: chứa các đỉnh đã xét.
- MO: chứa các đỉnh đang xét.

```
1. MO = ∅; MO = MO ∪ {T_O}
2.      while (MO != ∅)
        {
                n = get(MO) //lấy đỉnh đầu trong danh sách MO
                if (n == T_G) //nếu n là trạng thái kết thúc
                        return TRUE //tìm kiếm thành công, dừng
                DONG = DONG ∪ {n} //đánh dấu n đã được xét
                for các đỉnh kể v của n
                        if (v chưa được xét) //v chưa ở trong DONG
                                MO = MO ∪ {v} //đưa v vào cuối danh sách MO
                                Father(v) = n //lưu lại vết đường đi từ n đến v
        }
```

Chúng ta có một số nhận xét sau đây về thuật toán tìm kiếm theo chiều rộng:

- Trong tìm kiếm theo chiều rộng, trạng thái nào được sinh ra trước sẽ được phát triển trước, do đó danh sách MO được xử lý như hàng đợi. Trong bước 2, ta cần kiểm tra xem n có là trạng thái kết thúc hay không. Nói chung các trạng thái kết thúc được xác định bởi một số điều kiện nào đó, khi đó ta cần kiểm tra xem n có thỏa mãn các điều kiện đó hay không.

- Nếu bài toán có nghiệm (tồn tại đường đi từ trạng thái ban đầu tới trạng thái đích), thì thuật toán tìm kiếm theo chiều rộng sẽ tìm ra nghiệm, đồng thời đường đi tìm được sẽ là ngắn nhất. Trong trường hợp bài toán vô nghiệm và không gian trạng thái hữu hạn, thuật toán sẽ dừng và cho thông báo vô nghiệm.

Đánh giá tìm kiếm theo chiều rộng:

- Bây giờ ta đánh giá thời gian và bộ nhớ mà tìm kiếm theo chiều rộng đòi hỏi. Giả sử, mỗi trạng thái khi được phát triển sẽ sinh ra b trạng thái kề. Ta sẽ gọi b là nhân tố nhánh. Giả sử rằng, nghiệm của bài toán là đường đi có độ dài d. Bởi nhiều nghiệm có thể được tìm ra tại một đỉnh bất kỳ ở mức d của cây tìm kiếm, do đó số đỉnh cần xem xét để tìm ra nghiệm là:

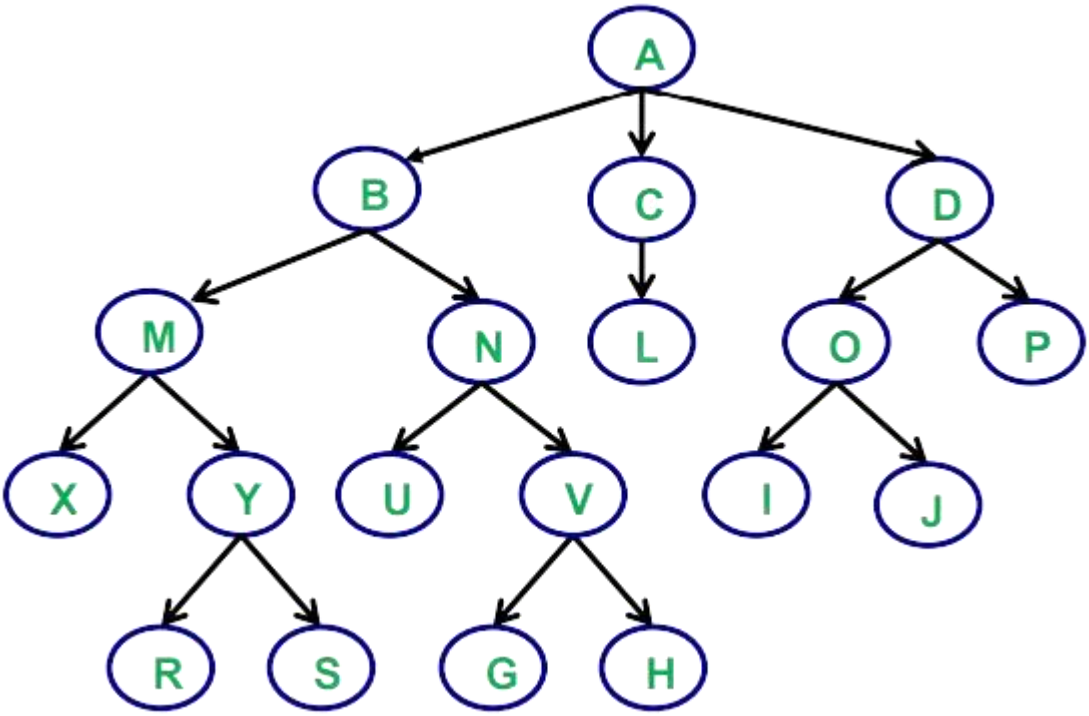
$1 + \mathrm{b} + b^2 + \ldots + b^{\mathrm{d}-1} + \mathrm{k}$

Trong đó k có thể là 1, 2, …, bd. Do đó số lớn nhất các đỉnh cần xem xét là: $1 + \mathrm{b} + b^2 + \ldots + b^{\mathrm{d}-1}$.

Như vậy, độ phức tạp thời gian của thuật toán tìm kiếm theo chiều rộng là $O(b^d)$.Độ phức tạp không gian cũng là $O(b^d)$, bởi vì ta cần lưu vào danh sách MO tất cả các đỉnh của cây tìm kiếm ở mức d, số các đỉnh này là b^d.

c. Ví dụ thuật toán tìm kiếm theo chiều rộng

Cho đồ thị như hình vẽ sau:



Hình 1.10. Đồ thị tìm kiếm

Đỉnh đầu T_O = A, T_G = {R}

Tìm đường đi p từ T_O đến T_G bằng phương pháp tìm kiếm theo chiều rộng?

n	B(n)	MO	DONG
		A	
A	B, C, D	B, C, D	A
B	M, N	C, D, M, N	A, B
C	L	D, M, N, L	A, B, C
D	O, P	M, N, L, O, P	A, B, C, D
M	X, Y	N, L, O, P, X, Y	A, B, C, D, M
N	→ là đích → là dừng		

Hình 1.11. Bảng duyệt theo BFS

Xây dựng đường đi có hành trình: p = A → B → N

Nhận xét:

- Nếu trong đồ thị tồn tại đường đi từ T_O đến 1 đỉnh $T_G \in$ Goal thì hàm BFS sẽ dừng lại và cho đường đi p có độ dài ngắn nhất.

- Với BFS các đỉnh được duyệt theo từng mức (theo chiều rộng).

- Thuật toán BFS có độ phức tạp O(b^d) với b là bậc của cây và d là chiều sâu của cây.

1.3.3. Tìm kiếm sâu dần

a. Tư tưởng của chiến lược tìm kiếm sân dần

- Xuất phát từ đỉnh ban đầu, thực hiện tìm kiếm sâu dần với giới hạn độ sâu tăng dần từ 0, 1, 2, …

- Mỗi lần lặp, chỉ xét các đỉnh đến giới hạn độ sâu cho trước, quá giới hạn sẽ quay lui.

- Quá trình kết thúc khi tìm thấy đỉnh T_G hoặc đã xét hết các độ sâu có thể.

b. Thuật toán tìm kiếm sâu dần

Vào: - Đồ thị G = (V, E)

- Đỉnh đầu T_0 và Goal chứa tập các đỉnh đích

- Độ sâu ds = k giới hạn độ sâu

Ra: Đường đi o: $T_0 \rightarrow T_G \in$ Goal

Phương pháp: //Sử dụng hai danh sách Closed và Open

//Closed hoạt động theo nguyên tắc FIFO

//Open vừa hoạt động theo nguyên tắc FIFO và vừa hoạt động theo LIFO

```
void IDS ()
{
    Open = {T0}, ds = k;
    While Open ≠ ∅ do
    {
        N ← get(Open)
        if (n = TG) then return True
        Closed = Closed ∪ {n}
        case d(n) do
        {
            0 ... ds – 1: Đặt A(n) vào đầu Open
            ds: Đặt A(n) vào cuối Open
            ds + 1:
```

{

ds = ds + k *//tăng độ sâu*

if k = 1 then đặt A(n) vào cuối tập Open

else đặt A(n) vào đầu tập Open

}

}

}

return True

}

c. Ví dụ thuật toán tìm kiếm sâu dần

Xét đồ thị như hình 1.9, đỉnh xuất phát từ T_0 = A, Goal = {R, O} và độ sâu k = 2.

n	d(n)	B(n)	MO	DONG
			A	
A	0	B, C, D	B, C, D	A
B	1	M, N	M, N, C, D	A, B
M	2	X, Y	N, C, D, X, Y	A, B, M
N	2	U, V	C, D, X, Y, U, V	A, B, M, N
C	1	L	L, D, X, Y, U, V	A, B, M, N, C
L	2	∅	D, X, Y, U, V	A, B, M, N, C, L
D	1	O, P	O, P, X, Y, U, V	A, B, M, N, C, L, D
O		→ là đích → là dừng		

Hình 1.12. Duyệt đồ thị theo tìm kiếm sâu dần

Đỉnh O ∈ Goal nên dừng quá trình tìm kiếm và xây dựng đường đi p. Đường đi này có hành trình là:

P = A → D → O

Kết quả: Nếu đồ thị G tồn tại đường đi p thì thủ tục tìm kiếm sâu dần sẽ dừng và cho đường đi p có độ dài khác độ dài đường đi ngắn nhất không quá k -1.

Nhận xét:

- Tìm kiếm sâu dần là kết hợp của tìm kiếm chiều sâu và tìm kiếm chiều rộng.

- Nếu k = 1 thì tìm kiếm sâu dần trở thành BFS.

- Nếu k là chiều cao của cây thì tìm kiếm sâu dần trở thành DFS.

- Thuật toán này cũng có độ phức tạp O(b^d).

1.3.4. So sánh thuật toán

	BFS	DFS
Thứ tự các đỉnh khi duyệt đồ thị	Các đỉnh được duyệt theo từng mức	Các đỉnh được duyệt theo từng nhánh
Độ dài đường đi p từ T_0 đến T_0	Ngắn nhất	Có thể không ngắn nhất
Tính hiệu quả	- Chiến lược có hiệu quả khi lời giải nằm ở mức thấp (gần gốc cây) - Thuận lợi khi tìm kiếm nhiều lời giải	- Chiến lược có hiệu quả khi lời giải nằm gần hướng đi được chọn theo phương án - Thuận lợi khi tìm kiếm lời giải
Sử dụng bộ nhớ	Lưu trữ toàn bộ không gian trạng thái của bài toán	Lưu trữ các trạng thái đang xét
Trường hợp tốt nhất	Vét cạn toàn bộ	Phương án chọn đường đi chính xác có lời giải trực tiếp
Trường hợp xấu nhất	Vét cạn	Vét cạn

Hình 1.13. Bảng so sánh 2 thuật toán DFS và BFS [5]

CHƯƠNG 3. XÂY DỰNG ỨNG DỤNG VÀ GIẢI QUYẾT BÀI TOÁN BẰNG THUẬT TOÁN TÌM KIẾM MÙ (BFS)

2.1. Phát biểu và mô tả bài toán

2.1.1. Phát biểu bài toán

Viết chương trình mô phỏng bài toán người lái đò. Bài toán phát biểu như sau: Tại bến sông nọ có bắp cải, sói và dê muốn bác lái đò chở qua sông. Biết rằng tại một thời điểm thuyền của bác lái đò chỉ chở tối đa được 2 khách. Nếu sói và dê đứng riêng với nhau (không có mặt bác lái đò và bắp cải) thì sói sẽ ăn thịt dê. Nếu dê và bắp cải đứng riêng với nhau (không có mặt bác lái đò và sói) thì dê sẽ ăn bắp cải [6].

2.1.2. Mô tả bài toán

Trong bài toàn người đò này, ta có bắp cải, sói, dê và người lái đò. Mục tiêu là di chuyển tất cả các đối tượng qua sông theo các ràng buộc sau:

- Người lái đò chỉ có thể chở tối đa 2 người 1 lần (bao gồm cả người lái đò).
- Nếu sói và dê đứng riêng với nhau (không có mặt bác lái đò và bắp cải) thì sói sẽ ăn thịt dê.
- Nếu dê và bắp cải đứng riêng với nhau (không có mặt người lái đò và sói) thì dê sẽ ăn bắp cải.

Hình 2.1. Ảnh minh họa bài toán người lái đò

2.2. Phân tích bài toán

2.2.1. Trạng thái

a. Đặt trạng thái

- Dùng hai cặp dấu ngoặc biểu thị 2 bên bờ sông.
- Bờ bên trái là bờ xuất phát (bờ đầu).
- Bờ bên phải là bờ bên kia sông (bờ đích).
- Các nhân vật trong bài toán là:
    + Người lái đò kí hiệu là "Người"
    + Sói kí hiệu là "Sói"
    + Dê kí hiệu là "Dê"
    + Bắp cải kí hiệu là "Bắp cải"

b. Trạng thái

- Trạng thái bài toán: (Dê, Sói, Người, Bắp cải) () với dê, sói, người, bắp cải tương ứng với các nhân vật trong bài toán trên bờ xuất phát.
- Trạng thái đầu: (Dê, Sói, Người, Bắp cải) ().
- Trạng thái đích: () (Người, Dê, Sói, Bắp cải).

2.2.2. Các toán tử dịch chuyển

a. Mô tả toán tử dịch chuyển

- Toán tử dịch chuyển 1: Người chèo thuyền từ bờ này sang bờ kia mà không chở theo gì.
- Toán tử dịch chuyển 2: Người chở Sói qua sông từ bờ này sang bờ kia.
- Toán tử dịch chuyển 3: Người chở Dê qua sông từ bờ này sang bờ kia.
- Toán tử dịch chuyển 4: Người chở Bắp cải qua sông từ bờ này sang bở kia.

b. Ký hiệu trạng thái

//Mô tả minh họa trạng thái

- Toán tử dịch chuyển 1: (Người, Sói, Dê, Bắp cải) () → (Sói, Dê, Bắp cải) (Người)
- Toán tử dịch chuyển 2: (Người, Sói, Dê, Bắp cải) () → (Dê, Bắp cải) (Người, Sói).
- Toán tử dịch chuyển 3: (Người, Sói, Dê, Bắp cải) () → (Sói, Bắp cải) (Người, Dê).
- Toán tử dịch chuyển 4: (Người, Sói, Dê, Bắp cải) () → (Sói, Dê) (Người, Bắp cải).

c. Điều kiện ràng buộc

- Chỉ Người có thể chèo thuyền.

- Mỗi lần Người chỉ có thể chở theo tối đa 1 vật (Sói, Dê, hoặc Bắp cải), hoặc đi một mình.

- Sau mỗi lần di chuyển, kiểm tra điều kiện an toàn (điều kiện đúng):

+ Không để Sói và Dê ở cùng bở mà không có Người.

+ Không để Dê và Bắp cải ở cùng bờ mà không có Người.

2.2.3. Vẽ không gian trạng thái

a) Cách biểu diễn đồ thị

Không gian trạng thái được biểu diễn bằng đồ thị vô hướng, trong đó:

- Đỉnh: các trạng thái hợp lệ (P,W,G,C)

- Cạnh: ứng với một hành động hợp lệ từ trạng thái này đến trạng thái khác

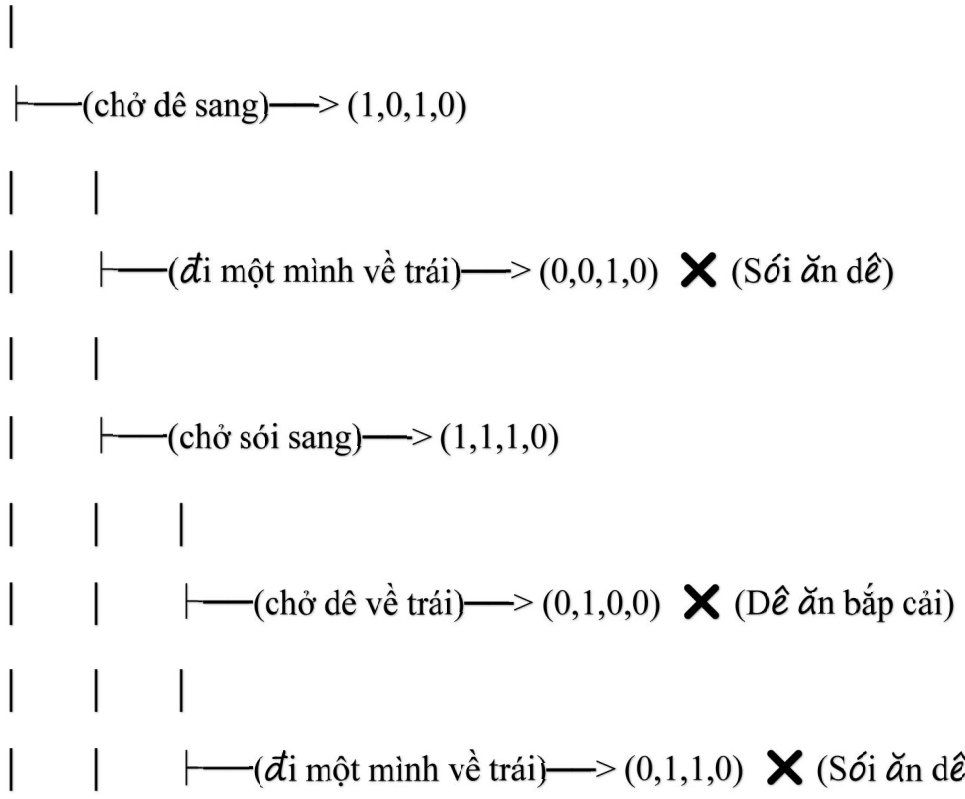
b) Ví dụ một phần không gian trạng thái

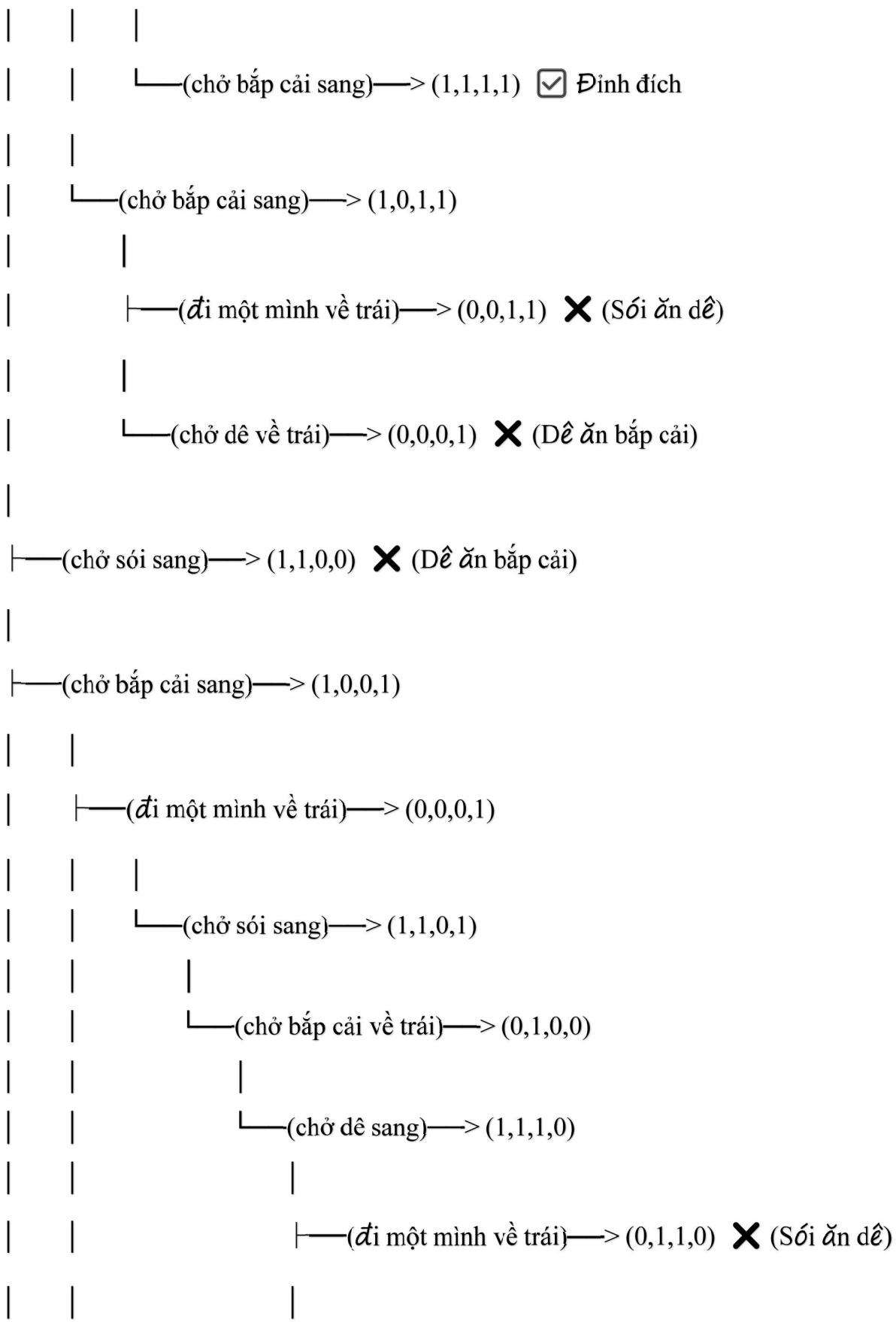
(0,0,0,0) --(chở dê) --> (1,0,1,0)

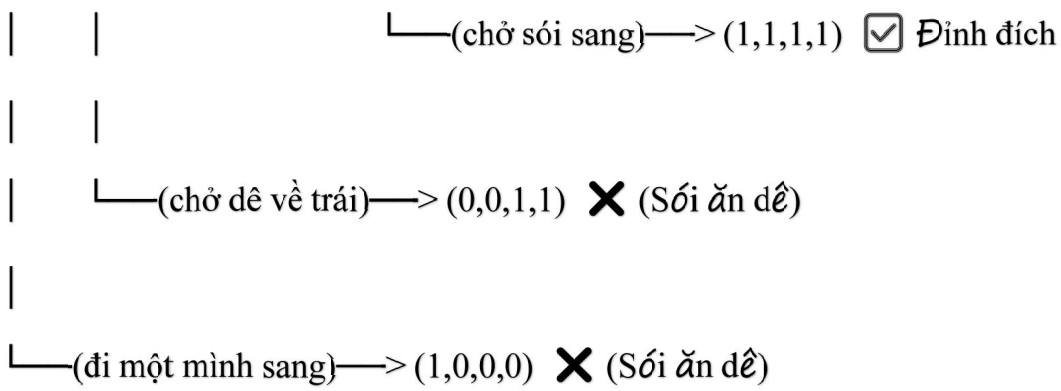
(1,0,1,0) --(đi một mình) --> (0,0,1,0) ✗ không hợp lệ (sói ăn dê)

c) Vẽ đồ thị trạng thái

Đỉnh đầu: (0,0,0,0) -- Tất cả bên bờ trái (Người lái đò, Sói, Dê, Bắp cải)




└──(chở bắp cải sang)──> (1,1,1,1) ☑ Đỉnh đích
└──(chở bắp cải sang)──> (1,0,1,1)
├──(đi một mình về trái)──> (0,0,1,1) ✖ (Sói ăn dê)
└──(chở dê về trái)──> (0,0,0,1) ✖ (Dê ăn bắp cải)
├──(chở sói sang)──> (1,1,0,0) ✖ (Dê ăn bắp cải)
├──(chở bắp cải sang)──> (1,0,0,1)
├──(đi một mình về trái)──> (0,0,0,1)
└──(chở sói sang)──> (1,1,0,1)
└──(chở bắp cải về trái)──> (0,1,0,0)
└──(chở dê sang)──> (1,1,1,0)
├──(đi một mình về trái)──> (0,1,1,0) ✖ (Sói ăn dê)



Biểu đồ trạng thái (theo tiêu chí Đỉnh - Cung - Đường đi):

2.2.4 Ví dụ minh họa một lời giải

(0, 0, 0, 0) - Trạng thái ban đầu (người lái đò, sói, dê, bắp cải đều ở bờ bên trái)

(1, 0, 1, 0) - Người lái đò và dê qua bờ bên phải

(0, 0, 1, 0) - Người lái đò quay lại bờ bên trái

(0, 1, 1, 1) - Người lái đò, sói và bắp cải qua bờ bên phải

(1, 1, 1, 0) - Người lái đò và dê quay lại bờ bên trái

(1, 1, 0, 0) - Người lái đò và dê qua bờ bên phải

(1, 0, 0, 0) - Người lái đò quay lại bờ bên trái

(1, 0, 0, 1) - Người lái đò và bắp cải qua bờ bên phải

(0, 0, 0, 1) - Người lái đò quay lại bờ bên trái

(0, 1, 0, 1) - Người lái đò và sói qua bờ bên phải

(1, 1, 0, 1) - Người lái đò quay lại bờ bên trái

(1, 1, 1, 1) - Người lái đò và dê qua bờ bên phải (trạng thái đích)

2.3. Xây dựng ứng dụng

2.3.1. Cài đặt cấu trúc trạng thái

- Mô tả vị trí của các nhân vật (Người, Dê, Sói, Bắp cải) ở bờ trái hoặc bờ phải của con sông tại

- Trạng thái ban đầu và trạng thái kết thúc được lưu lần lượt vào các biến là bat_dau và ket_thuc.

- Điều kiện trạng thái hợp lệ là khi không có sự cố Dê bị Sói ăn hoặc Dê ăn Bắp cải nếu không có Người trông coi

```python
from collections import deque

# Hàm kiểm tra trạng thái hợp lệ
def hop_le(trang_thai):
    nguoi, soi, de, bap_cai = trang_thai
    if soi == de and nguoi != de:
        return False
    if de == bap_cai and nguoi != de:
        return False
    return True
```

Hình 2.3. Khởi tạo trạng thái ban đầu và hàm kiểm tra trạng thái hợp lệ

2.3.2. Cài đặt cấu trúc để xây dựng câu tìm kiếm không gian trạng thái

Trong quá trình xây dựng giải thuật cho bài toán Người lái đò, bước quan trọng là xác định các trạng thái có thể xảy ra khi di chuyển qua sông. Đoạn mã trên thực hiện việc sinh trạng thái mới từ trạng thái hiện tại bằng cách giả lập các lần di chuyển của Người lái đò cùng với các đối tượng Sói, Dê, và Bắp cải. Mỗi trạng thái mới sẽ được kiểm tra để đảm bảo tính hợp lệ, tức là không có tình huống Dê bị Sói ăn hoặc Bắp cải bị Dê ăn khi vắng mặt Người lái đò. Việc sinh và kiểm tra trạng thái này đóng vai trò nền tảng trong việc áp dụng các thuật toán tìm kiếm như BFS để tìm ra chuỗi hành động tối ưu đưa toàn bộ các đối tượng qua sông an toàn.

- Sử dụng hàm sinh_trang_thai_moi_uu_tien_bap_cai(trang_thai) để sinh ra các trạng thái hợp lệ tiếp theo từ một trạng thái hiện tại trong bài toán Người Lái Đò.

- Đây là một đoạn mã Python để sinh các trạng thái mới từ trạng thái hiện tại trong bài toán Người lái đò đưa Sói, Dê và Bắp cải qua sông.

-Trang_thai là một tuple 4 phần tử:

(vị trí người, vị trí sói, vị trí dê, vị trí bắp cải)

Ví dụ: (0, 0, 1, 0) nghĩa là người, sói và bắp cải ở bờ trái (0), còn dê ở bờ phải (1).

-ket_qua: danh sách lưu các trạng thái mới hợp lệ sinh ra.

- Xác định vị trí của Người:

+ Người chỉ có thể chở đối tượng cùng bờ với mình.

+Ưu tiên chở theo thứ tự: bắp cải → dê → sói → đi một mình

- Lặp qua từng lựa chọn để sinh trạng thái mới

+ Người sang bờ đối diện (1 - nguoi), vì 0 → 1 hoặc 1 → 0.

Khởi tạo vị trí mới cho các đối tượng

Ban đầu, giữ nguyên vị trí các đối tượng khác.

Nếu người chở một đối tượng nào, thì đối tượng đó cũng đổi bờ.

Tạo trạng thái mới và kiểm tra hợp lệ

Tạo trạng thái mới sau khi người di chuyển (và có thể chở theo).

Dùng hàm hop_le(...) (đã định nghĩa trước đó) để kiểm tra xem trạng thái có vi phạm luật không:

Sói và dê ở cùng một bờ mà không có người → lỗi

Dê và bắp cải ở cùng một bờ mà không có người → lỗi

Nếu hợp lệ, thêm vào danh sách kết quả ket_qua.

- Kết quả trả về: Một danh sách các trạng thái hợp lệ có thể đạt được từ trạng thái hiện tại.

```python
# Sinh trạng thái mới, ưu tiên bắp cải
def sinh_trang_thai_moi_uu_tien_bap_cai(trang_thai):
    nguoi, soi, de, bap_cai = trang_thai
    ket_qua = []

    # Ưu tiên: bắp cải -> dê -> sói -> đi một mình
    lua_chon = []
    if nguoi == bap_cai:
        lua_chon.append('bắp cải')
    if nguoi == de:
        lua_chon.append('dê')
    if nguoi == soi:
        lua_chon.append('sói')
    lua_chon.append(None)  # đi một mình cuối cùng

    for doi_tuong in lua_chon:
        nguoi_moi = 1 - nguoi
        soi_moi, de_moi, bap_cai_moi = soi, de, bap_cai
        if doi_tuong == 'sói':
            soi_moi = 1 - soi
        elif doi_tuong == 'dê':
            de_moi = 1 - de
        elif doi_tuong == 'bắp cải':
            bap_cai_moi = 1 - bap_cai
        trang_thai_moi = (nguoi_moi, soi_moi, de_moi, bap_cai_moi)
        if hop_le(trang_thai_moi):
            ket_qua.append((trang_thai_moi, doi_tuong))
    return ket_qua
```

Hình 2.4. Cài đặt hàm xây dựng cây tìm kiếm không gian trạng thái

2.3.3. Cài đặt thuật toán BFS để tìm tất cả các đường đi

Trong quá trình triển khai bài toán Người lái đò, thuật toán tìm kiếm theo chiều rộng (BFS) đã được sử dụng để tìm ra tất cả các con đường hợp lệ từ trạng thái ban đầu, khi tất cả Người, Sói, Dê và Bắp cải cùng ở bờ trái, đến trạng thái đích khi tất cả đã qua bờ phải an toàn.

Thuật toán BFS duyệt qua từng trạng thái theo mức độ, đảm bảo tìm được tất cả các cách thức di chuyển thỏa mãn ràng buộc an toàn (không để xảy ra tình trạng Sói ăn Dê, hoặc Dê ăn Bắp cải khi vắng Người). Kết quả thực nghiệm cho thấy, BFS không chỉ

tìm được một lời giải mà còn liệt kê đầy đủ các phương án khả thi, giúp người giải bài toán có cái nhìn tổng quát hơn về các chiến lược di chuyển hợp lý trong bài toán này.

- Đây là đoạn mã triển khai thuật toán BFS (Breadth First Search) để tìm tất cả các đường đi từ trạng thái bắt đầu đến trạng thái đích trong bài toán Người lái đò.

- Đầu vào:

+ bat_dau: trạng thái xuất phát, ví dụ (0, 0, 0, 0) (mọi thứ ở bờ trái)

+ ket_thuc: trạng thái mong muốn, ví dụ (1, 1, 1, 1) (mọi thứ qua bờ phải).

- Khởi tạo queue:

+ Sử dụng hàng đợi deque để lưu trữ các trạng thái và đường đi tương ứng..

+ Mỗi phần tử là một bộ 3:

+ trang_thai hiện tại

+ duong_di: danh sách các trạng thái đã đi qua

+ hanh_dong: mô tả hành động ứng với từng bước path: Danh sách các trạng thái đã đi qua để đến trạng thái hiện tại.

+ da_tham: dùng để tránh lặp lại trạng thái đã duyệt.

- Duyệt BFS:

+ Kiểm tra trạng thái đích

+ Nếu đã tới đích → trả về:

+ sinh các trạng thái hợp lệ ưu tiên chở bắp cải.

+ Thêm trạng thái mới vào hàng đợi nếu chưa thăm .Tránh lặp trạng thái gây vô hạn hoặc sai logic.

+ Ghi nhận hành động và đưa vào hàng đợi

+Ghi chú hành động đang thực hiện:"Đi một mình" nếu không chở gì,"Đi cùng dê/sói/bắp cải" nếu có đối tượng đi cùng

+Cập nhật duong_di và hanh_dong tương ứng.

```python
# BFS ưu tiên bắp cải
def bfs_uu_tien_bap_cai(bat_dau, ket_thuc):
    hang_doi = deque()
    hang_doi.append((bat_dau, [], []))
    ket_qua = []
    da_tham = set()

    while hang_doi:
        hien_tai, duong_di, hanh_dong = hang_doi.popleft()
        if hien_tai == ket_thuc:
            ket_qua.append((duong_di + [hien_tai], hanh_dong))
            continue  # tìm thêm lời giải khác
        for trang_thai_moi, doi_tuong in sinh_trang_thai_moi_uu_tien_bap_cai(hien_tai):
            if trang_thai_moi not in duong_di:  # cho phép trạng thái xuất hiện nhiều lần ở nhánh khác
                ten_hanh_dong = "Đi một mình" if doi_tuong is None else f"Đi cùng {doi_tuong}"
                hang_doi.append((trang_thai_moi, duong_di + [hien_tai], hanh_dong + [ten_hanh_dong]))
    return ket_qua
```

Hình 2.5. Cài đặt thuật toán BFS để tìm tất cả đường đi hợp lệ

2.3.4. Cài đặt chương trình thuật toán và in kết quả

Sau khi cài đặt và thực thi thuật toán BFS để giải bài toán Người lái đò, chúng tôi đã thu được kết quả cụ thể là tìm ra tất cả các cách giải hợp lệ để đưa Người, Sói, Dê và Bắp cải qua sông an toàn. Chương trình đã liệt kê đầy đủ các bước di chuyển từ trạng thái khởi đầu đến trạng thái đích, đảm bảo không xảy ra trường hợp Dê bị Sói ăn hoặc Bắp cải bị Dê ăn khi vắng mặt Người lái đò.

- Đầu vào:

    + bat_dau: Trạng thái ban đầu (thường là tất cả ở bờ trái).

    + ket_thuc: Trạng thái đích (thường là tất cả qua bờ phải).

- Gọi hàm BFS: Hàm bfs_uu_tien_bap_cai(bat_dau,ket_thuc) sẽ trả về danh sách tất cả các đường đi hợp lệ từ trạng thái bắt đầu đến trạng thái đích.

```python
# Hiển thị trạng thái hai bờ
def in_ben(trang_thai):
    ben_trai = []
    ben_phai = []
    ten = ['người lái đò', 'sói', 'dê', 'bắp cải']
    for i in range(4):
        if trang_thai[i] == 0:
            ben_trai.append(ten[i])
        else:
            ben_phai.append(ten[i])
    return ben_trai, ben_phai

# --- CHẠY CHƯƠNG TRÌNH ---
bat_dau = (0, 0, 0, 0)
ket_thuc = (1, 1, 1, 1)
cac_loi_giai = bfs_uu_tien_bap_cai(bat_dau, ket_thuc)
if cac_loi_giai:
    print(f"Tìm được {len(cac_loi_giai)} lời giải:\n")
    for so, (loi_giai, hanh_dong) in enumerate(cac_loi_giai, 1):
        print(f"--- Lời giải {so} ---")
        for i in range(1, len(loi_giai)):
            trang_thai_truoc = loi_giai[i - 1]
            trang_thai_sau = loi_giai[i]
            hanh_dong_text = hanh_dong[i - 1]
            ben_trai, ben_phai = in_ben(trang_thai_truoc)
            print(f"Bước {i}:")
            print(f"\tBờ trái : {ben_trai}")
            print(f"\tBờ phải: {ben_phai}")
            print(f"\t Hành động: {trang_thai_truoc} --> {trang_thai_sau} -> {hanh_dong_text}\n")
        print(f" Hoàn thành sau {len(loi_giai) - 1} bước.\n")
else:
    print("Không tìm thấy lời giải.")
```

Hình 2.6. Cài đặt chương trình chạy và in kết quả bài toán

2.3.5. Kết quả bài toán Người lái đò ứng dụng thuật toán BFS

Trong quá trình giải quyết bài toán, Người lái đò bằng cách áp dụng thuật toán tìm kiếm theo chiều rộng (BFS -–Breadth First Search), chúng tôi đã thu được kết quả là tìm được dãy các trạng thái an toàn từ bờ khởi đầu đến bờ đích. Thuật toán BFS đã giúp đảm bảo tìm ra lời giải tối ưu với số lần di chuyển ít nhất để đưa toàn bộ hành khách (bao gồm Người lái đò, Sói, Dê và Bắp cải) sang bờ bên kia mà không xảy ra tình huống ăn thịt lẫn nhau.

Sau khi cài đặt và thực thi thuật toán BFS để giải bài toán Người lái đò, chúng tôi đã thu được kết quả là tìm ra 2 cách giải hợp lệ để đưa Người, Sói, Dê và Bắp cải qua sông an toàn. Các bước di chuyển cụ thể của từng cách đã được liệt kê rõ ràng như sau:

+ Cách 1: Người lái đò lần lượt thực hiện các bước đưa Dê, Sói và Bắp cải qua sông theo trình tự hợp lý, đảm bảo không xảy ra tình trạng Dê bị Sói ăn hoặc Bắp cải bị Dê ăn khi Người vắng mặt.

+ Cách 2: Một lộ trình khác cũng đưa tất cả qua sông an toàn, tuy có đôi chút khác biệt trong trình tự di chuyển, nhưng vẫn đảm bảo điều kiện an toàn trong suốt hành trình.

Kết quả này cho thế, với bài toàn tìm kiếm đường đi trong không gian trạng thái, thuật toán BFS đã phát huy hiệu quả khi tìm được toàn bộ các giải pháp có thể. Qua đó, người giải có thể linh hoạt lựa chọn lộ trình phù hợp nhất, đồng thời khẳng định rằng việc áp dụng tìm kiếm theo chiều rộng giúp đảm bảo không bỏ sót bất kỳ cách giải hợp lệ nào.

Hai lộ trình tìm được phản ánh tính đa dạng trong phương pháp giải quyết bài toán và minh chứng rằng, dù xuất phát từ cùng một trạng thái ban đầu, có thể tồn tại nhiều cách thức khác nhau để đạt được mục tiêu cuối cùng

```
Tìm được 2 lời giải:

--- Lời giải 1 ---
Bước 1:
    Bờ trái : ['người lái đò', 'sói', 'dê', 'bắp cải']
    Bờ phải: []
     Hành động: (0, 0, 0, 0) --> (1, 0, 1, 0) -> Đi cùng dê

Bước 2:
    Bờ trái : ['sói', 'bắp cải']
    Bờ phải: ['người lái đò', 'dê']
     Hành động: (1, 0, 1, 0) --> (0, 0, 1, 0) -> Đi một mình

Bước 3:
    Bờ trái : ['người lái đò', 'sói', 'bắp cải']
    Bờ phải: ['dê']
     Hành động: (0, 0, 1, 0) --> (1, 0, 1, 1) -> Đi cùng bắp cải

Bước 4:
    Bờ trái : ['sói']
    Bờ phải: ['người lái đò', 'dê', 'bắp cải']
     Hành động: (1, 0, 1, 1) --> (0, 0, 0, 1) -> Đi cùng dê

Bước 5:
    Bờ trái : ['người lái đò', 'sói', 'dê']
    Bờ phải: ['bắp cải']
     Hành động: (0, 0, 0, 1) --> (1, 1, 0, 1) -> Đi cùng sói

Bước 6:
    Bờ trái : ['dê']
    Bờ phải: ['người lái đò', 'sói', 'bắp cải']
     Hành động: (1, 1, 0, 1) --> (0, 1, 0, 1) -> Đi một mình

Bước 7:
    Bờ trái : ['người lái đò', 'dê']
    Bờ phải: ['sói', 'bắp cải']
     Hành động: (0, 1, 0, 1) --> (1, 1, 1, 1) -> Đi cùng dê

 Hoàn thành sau 7 bước.

--- Lời giải 2 ---
Bước 1:
    Bờ trái : ['người lái đò', 'sói', 'dê', 'bắp cải']
    Bờ phải: []
     Hành động: (0, 0, 0, 0) --> (1, 0, 1, 0) -> Đi cùng dê

Bước 2:
    Bờ trái : ['sói', 'bắp cải']
    Bờ phải: ['người lái đò', 'dê']
     Hành động: (1, 0, 1, 0) --> (0, 0, 1, 0) -> Đi một mình

Bước 3:
    Bờ trái : ['người lái đò', 'sói', 'bắp cải']
    Bờ phải: ['dê']
     Hành động: (0, 0, 1, 0) --> (1, 1, 1, 0) -> Đi cùng sói

Bước 4:
    Bờ trái : ['bắp cải']
    Bờ phải: ['người lái đò', 'sói', 'dê']
     Hành động: (1, 1, 1, 0) --> (0, 1, 0, 0) -> Đi cùng dê
```

```
Bước 5:
    Bờ trái : ['người lái đò', 'dê', 'bắp cải']
    Bờ phải: ['sói']
     Hành động: (0, 1, 0, 0) --> (1, 1, 0, 1) -> Đi cùng bắp cải

Bước 6:
    Bờ trái : ['dê']
    Bờ phải: ['người lái đò', 'sói', 'bắp cải']
     Hành động: (1, 1, 0, 1) --> (0, 1, 0, 1) -> Đi một mình

Bước 7:
    Bờ trái : ['người lái đò', 'dê']
    Bờ phải: ['sói', 'bắp cải']
     Hành động: (0, 1, 0, 1) --> (1, 1, 1, 1) -> Đi cùng dê

 Hoàn thành sau 7 bước.


Process finished with exit code 0
```

Hình 2.7. Kết quả bài toán Người lái đò ứng dụng thuật toán BFS

KẾT LUẬN

Trong bài tập lớn lần này, nhóm chúng em đã tiến hành nghiên cứu và triển khai ứng dụng các thuật toán tìm kiếm mù vào bài toán "Người lái đò". Mục tiêu chính của bài tập lớn là tìm hiểu các phương pháp tìm kiếm mù và áp dụng một trong các thuật toán này để giải quyết bài toán mang tính logic và trạng thái. Nhóm đã bắt đầu từ việc xây dựng mô hình toán học cho bài toán, xác định rõ các ràng buộc, quy tắc di chuyển và điều kiện hợp lệ giữa các trạng thái.

Chúng em đã tập trung nghiên cứu thuật toán tìm kiếm theo chiều rộng (BFS) và xây dựng chiến lược di chuyển thông minh nhằm tối ưu hóa quá trình tìm kiếm lời giải. Quá trình này giúp nhóm hiểu rõ hơn về không gian trạng thái, tư duy giải bài toán theo hướng thuật toán và khả năng phân tích vấn đề trong lĩnh vực Trí tuệ nhân tạo.

Qua bài báo cáo này, chúng em không chỉ củng cố thêm kiến thức về học phần Trí tuệ nhân tạo, đặc biệt là nhóm các thuật toán tìm kiếm mù, mà còn rút ra được nhiều kinh nghiệm quý giá từ việc thực hành và hiện thực hóa lời giải bằng lập trình. Đồng thời, bài tập lớn cũng giúp chúng em cải thiện các kỹ năng mềm như tra cứu tài liệu, khai thác thông tin trên Internet, làm việc nhóm, và đặc biệt là khả năng tiếp cận tài liệu chuyên ngành bằng tiếng Anh.Tuy nhiên, trong quá trình thực hiện, nhóm chúng em nhận thấy bản thân còn thiếu nhiều kiến thức và kinh nghiệm, đặc biệt là trong việc sử dụng ngôn ngữ lập trình Python, nên bài làm có thể còn tồn tại những thiếu sót hoặc sai sót. Chúng em rất mong thầy Trần Thanh Huân thông cảm và góp ý để nhóm có thể hoàn thiện hơn trong các bài tập sau.

Cuối cùng, nhóm xin chân thành cảm ơn sự hướng dẫn tận tình của ThS. Trần Thanh Huân, cùng với sự nỗ lực, đóng góp của tất cả các thành viên trong nhóm trong suốt quá trình thực hiện bài tập lớn này. Việc nghiên cứu và triển khai bài toán đã giúp chúng em không chỉ hiểu rõ hơn về cách giải quyết bài toán trạng thái mà còn rèn luyện được tư duy thuật toán, khả năng phân tích và kỹ năng lập trình – những yếu tố rất quan trọng trong hành trình học tập và làm việc sau này.

TÀI LIỆU THAM KHẢO

[1] N. P. Nga, Trí tuệ nhân tạo, Hà Nội: Nhà xuất bản Thống Kê, 2021.

[2] N. P. Nga, "Tìm kiếm trên không gian trạng thái," trong *Giáo trình Trí tuệ nhân tạo*, Hà Nội, Nhà xuất bản Thống Kê, 2021, trang. 25.

[3] P. N. Nga, "Hệ thống học kết hợp của Trường Đại Học Công Nghiệp Hà Nội," 30/12/2024. [Trực tuyến]. Địa chỉ: https://qlht.haui.edu.vn/pluginfile.php/1959463/mod_resource/content/2/%C4%90%E1%BB%81%20c%C6%B0%C6%A1ng%20b%C3%A0i%20gi%E1%BA%A3ng%20b%C3%A0i%203_TTNT_21052021.pdf. [Truy cập 15/2/2025].

[4] P. N. Nga, "Hệ thống học kết hợp của Trường Đại Học Công Nghiệp Hà Nội," 30/12/2024. [Trực tuyến]. Địa chỉ: https://qlht.haui.edu.vn/pluginfile.php/1959463/mod_resource/content/2/%C4%90%E1%BB%81%20c%C6%B0%C6%A1ng%20b%C3%A0i%20gi%E1%BA%A3ng%20b%C3%A0i%203_TTNT_21052021.pdf. [Truy cập 15/2/2025].

[5] T. Thơ, "Các thuật toán cơ bản trong AI," VIBLO, 13/1/2019. [Trực tuyến]. Địa chỉ: https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G. [Truy cập 15/2/2025].

[6] Đ. V. Thành, "Bài toán Người lái đò," 5/2023. [Trực tuyến]. Địa chỉ: https://www.studocu.vn/vn/document/truong-dai-hoc-cong-nghe-giao-thong-van-tai/tri-tue-nhan-tao/bai-toan-nguoi-lai-do/61185593. [Truy cập 15/2/2025].

[7] L. M. Hoàng and L. M. Hoàng, "Giải thuật và lập trình," VNOI WIKI, [Trực tuyến]. Địa chỉ: https://wiki.vnoi.info/algo/graph-theory/breadth-first-search.md. [Accessed 15/2/2025].

[8] N. P. Nga, "Hệ thống học kết hợp của Trường Đại Học Công Nghiệp Hà Nôi," [Trực tuyến]. Địa chỉ: https://qlht.haui.edu.vn/course/view.php?id=33445&section=3. [Truy cập 30/12/2024].