**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**===== o0o =====**



**BÀI TẬP LỚN MÔN: TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: TÌM HIỂU VỀ CÁC THUẬT TOÁN TÌM KIẾM MÙ VÀ ỨNG DỤNG VÀO BÀI TOÁN NGƯỜI LÁI ĐÒ**

*Thành viên nhóm: Nguyễn Xuân Dương - 2021600704*

*Nguyễn Quang Dương- 2021600653*

*Vũ Mạnh Hoàng - 2021600271*

*Lớp :*20231IT6043010 - K16

*Nhóm: 21*

***Giáo viên hướng dẫn:******Giảng viên. Nguyễn Lan Anh***

Hà Nội - 12/2023.

# **LỜI MỞ ĐẦU**

Để hoàn thành bản báo cáo này, chúng em đã nhận được rất nhiều sự hướng dẫn từ phía các thầy các cô trong khoa. Sự giảng dạy chu đáo, tận tình và sự giúp đỡ nhiệt tình từ các thầy các cô đã giúp chúng em hiểu ra nhiều vấn đề và hoàn thành bản báo cáo này tốt nhất.

Chúng em tỏ lòng biết ơn sâu sắc với cô Nguyễn Lan Anh, người cô đã tận tình hướng dẫn và giúp đỡ, chỉ bảo nhóm em trong suốt quá trình nghiên cứu đề tài và hoàn thành báo cáo này.

Sau khoảng thời gian cô Nguyễn Lan Anh ra đề tài, chúng em đã rất nỗ lực và cố gắng trong việc tìm hiểu về đề tài. Các bạn trong nhóm cùng các cộng sự đã rất chăm chỉ cũng như giúp đỡ lẫn nhau để cho ra một báo cáo hoàn hảo nhất đến thời điểm hiện tại. Một lần nữa nhóm em xin cảm ơn giảng viên Nguyên Lan Anh , các bạn trong lớp và tập thể nhóm làm việc đã cùng nhau hoàn thành tốt được bản báo cáo này.

**Chúng em xin chân thành cảm ơn!**

**MỤC LỤC**

[**LỜI MỞ ĐẦU 1**](#_gjdgxs)

[**DANH MỤC HÌNH ẢNH 3**](#_30j0zll)

[**DANH MỤC BẢNG BIỂU 4**](#_1fob9te)

[**CHƯƠNG I. GIỚI THIỆU CHUNG VỀ ĐỀ TÀI 6**](#_8yv8o7661qlk)

[**1.1. Tên đề tài 6**](#_vaxr38div70b)

[**1.2. Lý do chọn đề tài 6**](#_mkq1ebj1j64d)

[**1.3. Mục tiêu đề tài 6**](#_1472qaf7j2a1)

[**1.4. Phương pháp nghiên cứu 6**](#_jstkd6pfo5rd)

[**1.5. Đối tượng nghiên cứu 7**](#_rxi2wxtabigj)

[**1.6. Phạm vi nghiên cứu 7**](#_jv5ixwqhozgf)

[**CHƯƠNG II: KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM MÙ 7**](#_4wf6r66c0a2m)

[**2.1 Không gian trạng thái 7**](#_2et92p0)

[**2.2 Các thuật toán tìm kiếm mù 8**](#_tyjcwt)

[**2.2.1 Thuật toán tìm kiếm theo chiều sâu (Depth First Search) 8**](#_3dy6vkm)

[**2.2.2 Thuật toán tìm kiếm theo chiều rộng (Breadth first search) 10**](#_2s8eyo1)

[**2.3 Thuật toán tìm kiếm Heuristic 13**](#_26in1rg)

[**2.3.1. Tổng quát về thuật toán tìm kiếm heuristic 13**](#_ah6x9enynlga)

[**2.3.2 Thuật toán AT 15**](#_lnxbz9)

[**2.3.3 Thuật giải AKT 17**](#_44sinio)

[**2.3.4 Thuật giải A\* 20**](#_4i7ojhp)

[**2.4. Quá trình xử lý bài toán 27**](#_2bn6wsx)

[**2.4.1. Mô hình tổng quát 27**](#_mf9v9l5ourzr)

[**2.4.2. Quá trình xử lý bài toán bằng văn bản 27**](#_pjve7du7acv0)

[**CHƯƠNG III: XÂY DỰNG CHƯƠNG TRÌNH 28**](#_6o5zbj6mxosg)

[**3.1 Xây dựng chương trình thuật toán chiều sâu và chiều rộng 28**](#_598jnz7gdxi2)

[**3.1.1 Thuật toán tìm kiếm chiều sâu 28**](#_pililev258k9)

[**3.1.2 Thuật toán tìm kiếm chiều rộng 31**](#_jev5uojkaadk)

[**3.2 Mô tả bài toán người lái đò 32**](#_qsh70q)

[**3.2.1 Không gian trạng thái 33**](#_3as4poj)

[**3.2.2 Lời giải 34**](#_49x2ik5)

[**3.3 Cài đặt thuật toán 34**](#_3o7alnk)

[**KẾT LUẬN 39**](#_kg8jrgomzs07)

[**TÀI LIỆU THAM KHẢO**](#_6n86ug9j16uu) **40**

# **DANH MỤC HÌNH ẢNH**

Hình 2. 1: Lưu trạng thái từng đỉnh [25](#_23ckvvd)

Hình 2. 2: Khởi tạo các biến cần thiết [26](#_ihv636)

Hình 2. 3: Thuật toán DFS [26](#_32hioqz)

Hình 2. 4: Tìm đường đi [27](#_1hmsyys)

Hình 2. 5: Chạy thử chương trình [27](#_41mghml)

# **DANH MỤC BẢNG BIỂU**

Bảng 1. 1: Ví dụ thuật giải tìm kiếm theo chiều sâu [8](#_4d34og8)

Bảng 1. 2: Ví dụ thuật giải tìm kiếm theo chiều rộng [10](#_3rdcrjn)

Bảng 1. 3: Ví dụ thuật giải AT [13](#_1ksv4uv)

Bảng 1. 4: So sánh thuật toán DFS và BFS [22](#_3whwml4)

[Biểu đồ 1. 1: Ví dụ thuật toán tìm kiếm theo chiều sâu 7](#_1t3h5sf)

[Biểu đồ 1. 2: Ví dụ thuật toán tìm kiếm theo chiều rộng 10](#_17dp8vu)

[Biểu đồ 1. 3: Ví dụ thuật toán AT 12](#_35nkun2)

[Biểu đồ 1. 4: Ví dụ thuật toán AKT 16](#_1y810tw)

[Biểu đồ 1. 5: Ví dụ thuật toán A\* 19](#_2xcytpi)

Biều đồ 2. 1: Không gian trạng thái bài toán người lái đò [24](#_1pxezwc)

Biều đồ 2. 2: Đồ thị bài toán người lái đò [25](#_147n2zr)

# 

# **CHƯƠNG I. GIỚI THIỆU CHUNG VỀ ĐỀ TÀI**

## **1.1. Tên đề tài**

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

## **1.2. Lý do chọn đề tài**

Lựa chọn đề tài ứng dụng thuật toán tìm kiếm mù trong bài toán người lái đò là một quyết định có ý nghĩa lớn do nó mang lại nhiều lợi ích đáng kể. Trong bối cảnh người lái đò phải đối mặt với môi trường biển động và đầy thách thức, việc tích hợp thuật toán tìm kiếm mù có thể cải thiện đáng kể hiệu suất và an toàn trong quá trình điều khiển.

Ngoài ra, việc ứng dụng thuật toán tìm kiếm mù cũng có thể tăng cường khả năng dự đoán và tránh các tình huống nguy hiểm. Thông qua việc phân tích dữ liệu từ các cảm biến và hệ thống đo đạc, thuật toán có thể dự đoán trước những biến động có thể xảy ra, giúp người lái đò thực hiện các biện pháp phòng ngừa hiệu quả.

Tổng cộng, sự kết hợp giữa ngành công nghiệp đò và thuật toán tìm kiếm mù mang lại tiềm năng đột phá về hiệu suất và an toàn. Việc nghiên cứu và phát triển trong lĩnh vực này có thể đóng góp tích cực vào việc làm cho việc điều khiển đò trở nên linh hoạt, an toàn hơn và hiệu quả hơn trong môi trường biển động và khó khăn.

## **1.3. Mục tiêu đề tài**

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

## **1.4. Phương pháp nghiên cứu**

Phương pháp nghiên cứu tài liệu: Trên internet và giáo trình

## **1.5. Đối tượng nghiên cứu**

Thuật toán tìm kiếm

## **1.6. Phạm vi nghiên cứu**

Nghiên cứu về các thuật toán tìm kiếm mù và ứng dụng vào bài toán người lái đò

# **CHƯƠNG II: KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM MÙ**

## **2.1 Không gian trạng thái**

Không gian trạng thái là tập hợp tất cả các trạng thái của bài toán ứng với một cấu trúc biểu diễn nào đó. Một không gian trạng thái (state space) là 1 bộ [N, A, S, GD] trong đó:

N (node) là các nút hay các trạng thái của đồ thị.

A (arc) là tập cung (hay các liên kết) giữa các nút.

S (solution) là một tập chứa các trạng thái đích của bài toán ( S c N ^ S !0)

Các trạng thái trong GD (Goal Description) được mô tả theo một trong hai đặc tính:

+ Đặc tính có thể đo lường được các trạng thái gặp trong quá trình tìm kiếm. VD: Tic-tac-toe, 8-puzzle,...

+ Đặc tính của đường đi hình thành trong quá trình tìm kiếm. VD: TSP

Đường đi của lời giải (solution path) là một con đường đi qua đồ thị này từ một nút thuộc S đến một nút thuộc GD.

## **2.2 Các thuật toán tìm kiếm mù**

### **2.2.1 Thuật toán tìm kiếm theo chiều sâu (Depth First Search)**

**a, Khái niệm**

**Tìm kiếm ưu tiên chiều sâu** hay **tìm kiếm theo chiều sâu** (tiếng Anh: *Depth-first search - DFS*) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Thông thường, DFS là một dạng tìm kiếm thông tin không đầy đủ mà quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước. Trong dạng không đệ quy, tất cả các đỉnh chờ được phát triển được bổ sung vào một ngăn xếp LIFO.

Độ phức tạp không gian của DFS thấp hơn của BFS (tìm kiếm theo chiều rộng). Độ phức tạp thời gian của hai thuật toán là tương đương nhau và bằng O(|V| + |E|).

**b, 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 TG thì quay lui tới một đỉnh nào đó để sang nhánh khác.

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

**c, Thuật toán tìm kiếm theo chiều sâu.**

Lưu trữ: 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 = Ø; MO = MO ∪ {T0}
  2. while (MO != Ø)

{ n = get(MO) *// lấy đỉnh đầu trong danh sach 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 đc xét) *//v chưa ở trong DONG*

MO = MO ∪ {v} *//đưa v vào đầu DS MO*

father(v)=n*// lưu lại vết đường đi từ n đến v*

*}*

**d, Ví dụ thuật toán theo chiều sâu**

Cho đồ thị như hình vẽ sau:

A diagram of a diagram

Description automatically generated

*Biểu đồ 1. 1: Ví dụ thuật toán tìm kiếm theo chiều sâu*

Đỉnh đầu T0=A, TG = {R}

Tìm đường đi p từ To đến TG 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 | Ø | Y, N, C, D | A,B,M,X |
| Y | R, S | R, S, N, C, D | A,B,M,X,Y |
| R | là đích dừng |  |  |

*Bảng 1. 1: Ví dụ thuật giải tìm kiếm theo chiều sâu*

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 T0 đến 1 đỉnh TG  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 phức tạp O(bd) với b là bậc của cây và d là chiều sâu của cây. Tuy nhiên trong trường hợp xấu nhất cũng là O(bd).

### **2.2.2 Thuật toán tìm kiếm theo chiều rộng (Breadth first search)**

**a, Khái niệm**

Trong lý thuyết đồ thị, **tìm kiếm theo chiều rộng** (**BFS**) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại. Xem thêm thuật toán tìm kiếm theo chiều sâu, trong đó cũng sử dụng 2 thao tác trên nhưng có trình tự quan sát các đỉnh khác với thuật toán tìm kiếm theo chiều rộng.

Đây là một thuật toán trong trí tuệ nhân tạo. Cấu trúc dữ liệu được sử dụng là hàng đợi (queue).

**b, 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 TG hoặc đã hết các đỉnh để duyệt.

**c,Thuật toán tìm kiếm theo chiều rộng**

Lưu trữ: 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.

MO = Ø; MO = MO ∪ {T0}

while (MO != Ø)

{ n = get(MO) // lấy đỉnh đầu trong danh sach 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 đc xét) //v chưa ở trong DONG

MO = MO ∪ {v} //đưa v vào cuối DS 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 MỞ 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 + b + b2 +… + bd-1 + k

Trong đó k có thể là 1, 2, …, bd. Do đó số lớn nhất các đỉnh cần xem xét là: 1 + b + b2 +… + bd-1 .

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

C, Ví dụ thuật toán tìm kiếm theo chiều rộng:

Cho đồ thị như hình vẽ sau:

A diagram of a diagram

Description automatically generated

*Biểu đồ 1. 2: Ví dụ thuật toán tìm kiếm theo chiều rộng*

Đỉnh đầu T0=A, TG= {N}. Tìm đường đi p từ To đến TG 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 | dừng |  |

*Bảng 1. 2: Ví dụ thuật giải tìm kiếm theo chiều rộng*

Xây dựng đường đi có hành trình: p = A -> B -> N.

Nhận xét:

+ Nếu trong đồ thị tồn tại đường đi từ T0 đến 1 đỉnh TG -> 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(bd) với b là bậc của cây và d là chiều sâu của cây.

## **2.3 Thuật toán tìm kiếm Heuristic**

### **2.3.1. Tổng quát về thuật toán tìm kiếm heuristic**

**Heuristic** là các kỹ thuật dựa trên kinh nghiệm để giải quyết vấn đề, học hỏi hay khám phá nhằm đưa ra một giải pháp mà không được đảm bảo là tối ưu. Với việc nghiên cứu khảo sát không có tính thực tế, các phương pháp heuristic được dùng nhằm tăng nhanh quá trình tìm kiếm với các giải pháp hợp lý thông qua các suy nghĩ rút gọn để giảm bớt việc nhận thức vấn đề khi đưa ra quyết định. Ví dụ của phương pháp này bao gồm sử dụng một luật ngón tay cái, giả thuyết, phán đoán trực giác, khuôn mẫu hay nhận thức thông thường.

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau:

*Thường* tìm được lời giải tốt (nhưng không chắc là lời giải tốt nhất)

Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.

Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ sở như sau:

***Nguyên lý vét cạn thông minh:***

Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.

***Nguyên lý tham lam (Greedy):***

Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.

***Nguyên lý thứ tự:***

Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.

***Hàm Heuristic:***

Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các *hàm Heuristic*. Đó là các hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.

### **2.3.2 Thuật toán AT**

a, Khái niệm:

Thuật giải AT là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g (trong chiều dài thực sự của đường đi từ đỉnh bắt đầu đến đỉnh hiện tại).

Cho đồ thị G = (V, E) với V: tập đỉnh; E: Tập cung. Với mỗi một cung người ta gắn thêm một đại lượng được gọi là giá của cung.

C: E → R+

e ↦C(e)

Vấn đề đặt ra là tìm đường đi p tù T0 đến đỉnh T(G) ∈ Goal  sao cho tổng đường đi là nhỏ nhất:

Vào:

- Đồ thị G = (V, E)

C: E → R+

e ↦C(e)

- Đỉnh đầu T0 và Goal chứa tập các đỉnh đích.

Ra:

Đường đi p: T0 → TG ∈ Goal sao cho:

C(p) = g(nk) = min {g(n)/n ∈ Goal}.

b, Phương pháp sử dụng hai danh sách CLOSE và OPEN

void AT()

{

OPEN ={T0}, g(T0) = 0, CLOSE = ⊘

g(T0) = 0, tính(T0), f(T0) = g(T0) + h(T0)

while (OPEN ≠ ⊘)

{

n ← getNew (OPEN)

if (n = TG) then return path T0 → TG

else

{

for each m ∈ A(n) do

if (m ≠ OPEN + CLOSE) then

{

tính h(m), g(m) f(m) = g(m) +h(m)

cha(m) = n

OPEN = OPEN ∪ {m}

}

else {

g(m) = min{g(m), gnew(m)

CLOSE= CLOSE ∪ {n}

 }

}

return False;

}

C, Ví dụ thuật toán AT

Cho đồ thị (hình 3). Đỉnh xuất phát A và Goal = {D, H}

A diagram of a triangle

Description automatically generated

*Biểu đồ 1. 3: Ví dụ thuật toán AT*

| **n** | **A(n)** | **OPEN** | **CLOSE** |
| --- | --- | --- | --- |
|  |  | A(0) |  |
| A | B, C, E | B(2),C(4), F(6) | A |
| B |  | C(4), F(6) | A, B |
| C | D, E | D(12),E(6),F(6) | A, B, C |
| E |  | D(12), F(6) | A, B, C, E |
| F | G, H | G(11),H(7),D(12) | A, B, C, E, F |
| H | →là đích | → dừng |  |

*Bảng 1. 3: Ví dụ thuật giải AT*

Thuật giải AT kết thúc khi gặp H∈Goal và tìm được đường đi p=A->F->H có chi phí C(p)=7.

Kết quả: Nếu trong đồ thị G tồn tại đường đi p: T0 → TG Goal thì thủ tục AT sẽ dừng và cho kết quả đường đi có độ dài ngắn nhất.

Nhận xét:

- Nếu C(a)=1 ∀ a∈E thì AT trở thành BFS

- Nếu thay điều kiện g(n)->min bằng điều kiện d(n)->max, trong đó d(n) là độ sâu hiện tại của n thì AT trở thành DFS.

### **2.3.3 Thuật giải AKT**

a, Định nghĩa:

Với thuật giải AT trong quá trình TK chỉ xét đến các đỉnh và giá của chúng.

• Việc tìm đỉnh triển vọng phụ thuộc vào hàm g(n) là thông tin quá khứ.

• Giải thuật này không phù hợp với các bài toán có độ phức tạp hàm mũ (do phải xét trên số lượng lớn các đỉnh).

• Để khắc phục nhược điểm này người ta sử dụng thêm các thông tin bổ sung xuất phát từ bản thân bài toán để tìm ra các đỉnh triển vọng.

• Đường đi tối ưu sẽ tập trung xung quanh đường đi tốt nhất nếu sử dụng các thông tin đặc tả về bài toán (thông tin tương lai).

Theo thuật giải này, chi phí của đỉnh được xác định: f(n) = g(n)+h(n).

• Đỉnh n được chọn nếu f(n) đạt min.

• Việc lựa chọn hàm ước lượng h(n) dựa theo:

• Việc lựa chọn toán tử dựng cung sao cho có thể loại bớt các đỉnh không liên quan và tìm ra các đỉnh triển vọng.

• Sử dụng thêm các thông tin bổ sung nhằm xây dựng tập MO và cách lấy các đỉnh trong tập MO.

• Xây dựng hàm đánh giá h(n):

• Dựa vào xác suất của đỉnh trên đường đi tối ưu.

• Dựa vào khoảng cách, sự sai khác của các trạng thái đang xét với trạng thái đích hoặc các thông tin liên quan đến trạng thái đích.

b, Thuật toán

Vào:

- Đồ thị G= f: V->R+ (hàm chi phí) Đỉnh đầu T0 và đỉnh TG ∈ Goal.

Ra:

- Đường đi p: T0 ->TG

Phương pháp: Sử dụng 2 danh sách DONG và MO.

void AKT()

{

MO = {T0}, g(T0) =0;

Tính h(T0), f(T0) = g(T0)+h(T0)

 while (MO! = ⊘){

n = getnew (MO)

 if (n==TG) return TRUE;

else {

for (m ∈ B(n))

{

g(m)=g(n)+Cost(m,n);

Tính h(m), f(m)=g(m)+h(m);

MO=MO ∪ {m};

}

}

return FALSE;

}

C, Ví dụ thuật toán AKT

Chọn hàm f(n) = g(n) + h(n)

Trong đó: h(n) thông tin liên quan đến số đĩa ở cọc 3

g(n) là số lần chuyển đĩa từ T0 đến trạng thái n.

* Nếu cọc 3 chưa có đĩa nào thì h = 2
* Nếu cọc 3 có 1 đĩa nhỏ thì h = 3
* Nếu cọc 3 có 1 đĩa to thì h = 1
* Nếu cọc 3 có 2 đĩa và đĩa nhỏ ở trên đĩa to thì h = 0

A diagram of a network

Description automatically generated

*Biểu đồ 1. 4: Ví dụ thuật toán AKT*

### **2.3.4 Thuật giải A\***

**a, khái niệm**

A\* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị. Thuật toán này tìm một đường đi từ một [nút](https://vi.wikipedia.org/w/index.php?title=N%C3%BAt_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)&action=edit&redlink=1) khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này.

Ý tưởng trực quan

Xét bài toán tìm đường - bài toán mà A\* thường được dùng để giải. A\* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin (*informed tìm kiếm thuật toán*), nó chỉ xây dựng các tuyến đường "có vẻ" dẫn về phía đích.

Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A\* sử dụng một "đánh giá heuristic" về khoảng cách từ điểm bất kỳ cho trước tới đích. Trong trường hợp tìm đường đi, đánh giá này có thể là khoảng cách đường chim bay - một đánh giá xấp xỉ thường dùng cho khoảng cách của đường giao thông.

Điểm khác biệt của A\* đối với tìm kiếm theo lựa chọn tốt nhất là nó còn tính đến khoảng cách đã đi qua. Điều đó làm cho A\* "đầy đủ" và "tối ưu", nghĩa là, A\* sẽ luôn luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như vậy. A\* không đảm bảo sẽ chạy nhanh hơn các thuật toán tìm kiếm đơn giản hơn. Trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay lại. Trong trường hợp đó, việc thử các nút theo thứ tự "gần đích hơn thì được thử trước" có thể gây tốn thời gian.

**b, Thuật toán**

• A\* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp KGTT là một đồ thị.

• A\* sử dụng tập DONG để lưu những TT đã xét.

• A\* mở rộng AKT với việc bổ sung cách giải quyết trường hợp khi xét 1 đỉnh mà nó đã có trong tập MO hoặc DONG.

• Khi xét TT Ti, ngoài việc lưu 3 giá trị f, g, h để phản ánh chi phí của TT đó, A\* có lưu 2 thông số:

• TT cha của Ti (Cha(Ti)), nếu Ti có nhiều Cha thì chọn Cha(Ti) sao cho Cost(T0->Ti) đạt min.

• Danh sách TT kế tiếp của Ti: các TT Tk sao cho Cost(T0- >Tk) thông qua Ti đạt min.

void Asao(){

MO = {T0}, DONG=, g(T0)=0, Tính h(T0), f(T0)=g(T0)+h(T0);

while (MO!=){

n=getnew(MO)//lấy đỉnh n sao cho f(n) đạt min.

if (n==TG) return TRUE;

else {

for (mB(n)) {

if (m(MODONG)){

Tính h(m), g(m), f(m)=g(m)+h(m);

Cha(m)=n; MO=MO{m};

}else{

g(m) = min{gold(m),gnew(m)};

Cập nhật lại MO;

}

}

}

DONG = DONG{n};

}

return FALSE;

}

**c, Các tính chất**

Cũng như [tìm kiếm theo chiều rộng](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng) (*breadth-first search*), A\* là thuật toán *đầy đủ* (*complete*) theo nghĩa rằng nó sẽ luôn luôn tìm thấy một lời giải nếu bài toán có lời giải.

Nếu hàm heuristic  *h* có tính chất *thu nạp được* (*admissible*), nghĩa là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A\* có tính chất thu nạp được (hay *tối ưu*) *nếu sử dụng một tập đóng*. Nếu không sử dụng tập đóng thì hàm *h* phải có tính chất *đơn điệu* (hay *nhất quán*) thì A\* mới có tính chất tối ưu. Nghĩa là nó không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực. Phát biểu một cách hình thức, với mọi nút *x,y* trong đó *y* là nút tiếp theo của *x:*

h(x) ≤ g(y) – g(x) + h(y)

A\* còn có tính chất *hiệu quả một cách tối ưu* (*optimally efficient*) với mọi hàm heuristic , có nghĩa là không có thuật toán nào cũng sử dụng hàm heuristic đó mà chỉ phải mở rộng ít nút hơn A\*, trừ khi có một số lời giải chưa đầy đủ mà tại đó  dự đoán chính xác chi phí của đường đi tối ưu.

**d, Ví dụ**

Trạng thái ban đầu T0 = A, trạng thái đoch Goal = {B}, các số ghi cạnh các cung là độ dài đường đi, các số cạnh các đỉnh là giá trị của hàm h

A diagram of a network

Description automatically generated

*Biểu đồ 1. 5: Ví dụ thuật toán A\**

Ban đầu OPEN = {A, g(A) = 0, f(A) = 14}

Phát triển đỉnh A sinh ra các đỉnh con C, D, E và F. Tính giá trị của hàm f tại các đỉnh này ta có:

OPEN = {

g(C) = 9, f(C) = 9 + 15 = 24, cha(C) = A, g(D) = 7,

f(D) = 7 +6 = 13, cha(D) = A,

g(E) = 13, f(E) = 13 + 8 = 21, cha(E) = A, g(F) = 20,

f(F) = 20 +7 = 27, cha(F) = A

}

CLOSE = {A, g(A) = 0, f(A) = 14}

Do f(D) = 13 nho nhất nên chọn D để phát triển. Phát triển D, ta nhận được các đỉnh kế tiếp H và E.

g(H) = g(D) + cost(D, H) = 7 + 8 = 15, f(H) = 15 + 10 = 25. g(E) = g(D) + cost(D, E) = 7 + 4 = 11, f(E) = 11 + 8 =19.

Bây giờ ta bo sung hai đỉnh mới này vào tập OPEN. Tuy nhiên trong tập OPEN lúc này đã có đỉnh E nên cần phải so sánh g(E) đã có và g(E) vùa tonh được. Ta so sánh hai giá trị này và giữ lại giá trị nho hơn. Do vậy

OPEN = {g(C) = 9, f(C) = 9 + 15 = 24, cha(C) = A, g(E) = 11, f(E) = 11 + 8= 19, cha(E) = D, g(F) = 20, f(F) = 20 +7 = 27, cha(F) = A,

g(H) = 15, f(H) = 15 + 10 = 25, cha(H) = D

}

CLOSE = {A, g(A) = 0, f(A) = 14,

D, g(D) = 7, f(D) = 7 + 6 = 13, cha(D) = A

}

Với lập luận tương tự như trên ta chọn đỉnh E để phát triển. Các đỉnh kế tiếp của E là K và I

OPEN = {g(C) = 9, f(C) = 9 + 15 = 24, cha(C) = A, g(F) = 20, f(F) = 20 +7 = 27, cha(F) = A

g(H) = 15, f(H) = 15 + 10 = 25, cha(H) = D,

g(K) = 17, f(K) = 17 + 2 = 19, cha(K) = E,

g(I) = 19, f(I) = 19 + 4 = 23, cha(I) = E

}

CLOSE = {A, g(A) = 0, f(A) = 14,

D, g(D) = 7, f(D) = 7 + 6 = 13, cha(D) = A E, g(E) = 11, f(E) = 11 + 8 = 19, cha(E) = D

}

Chọn đỉnh K để phát triển. Các đỉnh tiếp của K là B. OPEN = {g(C) = 9, f(C) = 9 + 15 = 24, cha(C) = A,

g(F) = 20, f(F) = 20 +7 = 27, cha(F) = A

g(H) = 15, f(H) = 15 + 10 = 25, cha(H) = D

g(I) = 19, f(I) = 19 + 2 = 23, cha(I) = E,

g(B) = 23, f(B) = 23 + 0 = 23, cha(B) = K,

} CLOSE= {

A, g(A) = 0, f(A) = 14,

D, g(D) = 7, f(D) = 7 + 6 = 13, cha(D) = A,

E, g(E) = 11, f(E) = 11 + 8 = 19, cha(E) = D,

g(K) = 17, f(K) = 17 + 2 = 19, cha(K) = E

}

Trong tập OPEN có f(B) = f(I) nên chọn ngẫu nhiên một trong hai đỉnh này.

Giả sử chọn đỉnh B để phát triển.

Do B ∈ Goal nên quá trình tìm kiếm kết thúc. Để đưa ra đường đi ta try ngược lại trong tập CLOSE. Khi đó đường đi tìm được có chi phí c(p)= 23 và trình tự các đỉnh.

Nhận xét:

p: A → D → E → K → B

Đường đi tìm được có thể không phải là tốt nhất.

Nếu h(n) = 0 trong mọi trường hợp thì A\* trở thành AT.

Bảng so sánh 2 thuật toán DFS và DFS.

|  | 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 T0 đến TG | Ngắc 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 1 lời giải |
| Sử dụng bộ nhớ | Lưu trữ toàn bộ KGTT | Lưu trữ các TT 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 |

*Bảng 1. 4: So sánh thuật toán DFS và BFS*

## **2.4. Quá trình xử lý bài toán**

### **2.4.1. Mô hình tổng quát**

## 

### **2.4.2. Quá trình xử lý bài toán bằng văn bản**

Áp dụng cho bài toán người lái đò:

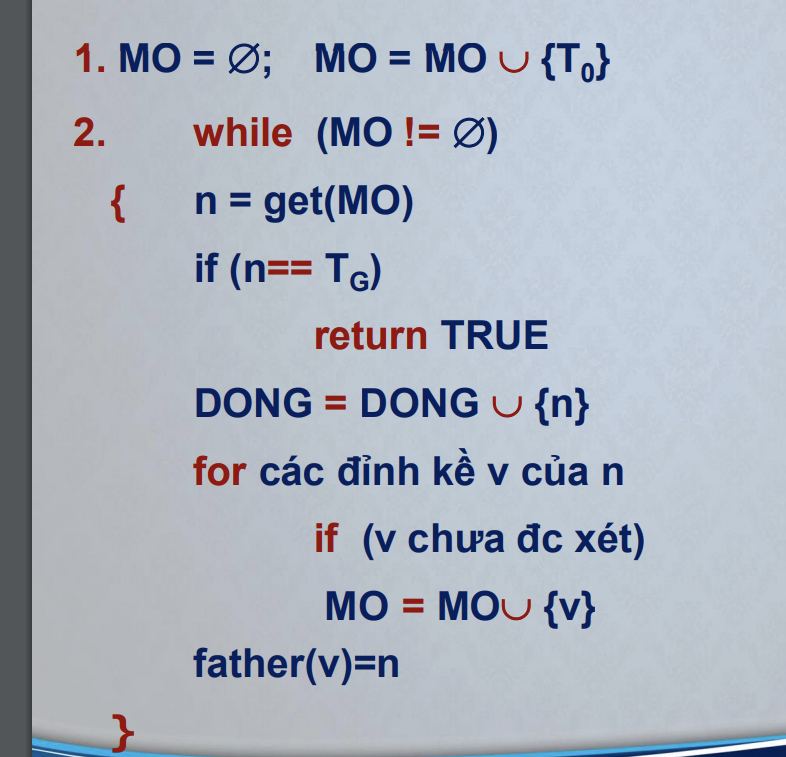
* Bước 1: Khởi tạo trạng thái ban đầu với tất cả các đối tượng đều ở bên xuất phát và không có ai ở trên thuyền
* Bước 2: Đưa trạng thái ban đầu vào hàng đợi và đánh dấu trạng thái ban đầu
* Bước 3: Lặp cho đến khi hàng đợi trống
* Bước 4: Lấy một trạng thái từ hàng đợi
* Bước 5: Kiểm tra xem trạng thái hiện tại có là trạng thái kết thúc hay không. Nếu đúng thì kết thúc thuật toán và trả về kết quả
* Bước 6: Nếu trạng thái hiện tại không phải là trạng thái kết thúc, thực hiện tất cả các hành động có thể từ trạng thái hiện tại bằng cách thay đổi vị trí các đối tượng trên bến và trên thuyền.
* Bước 7: Với mỗi hành động có thể, kiểm tra tính hợp lệ của bài toán đó dựa trên các ràng buộc của bài toán.
* Bước 8: Nếu hành động hợp lệ, tạo ra trạng thái mới bằng cách thực hiện hành động đó và thêm hành động mới vào hàng đợi.
* Bước 9: Đánh dấu trạng thái mới đã được tạo ra để tránh lặp lại với các trạng thái trước đó.
* Bước 10: Lặp lại từ bước 4 đến bước 9 cho đến khi tìm được trạng thái kết thúc hoặc hàng đợi trốn

## **CHƯƠNG III: XÂY DỰNG CHƯƠNG TRÌNH**

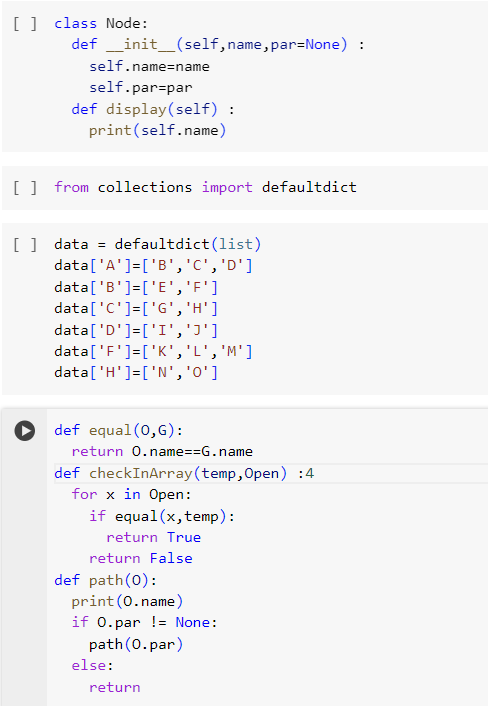
## **3.1 Xây dựng chương trình thuật toán chiều sâu và chiều rộng**

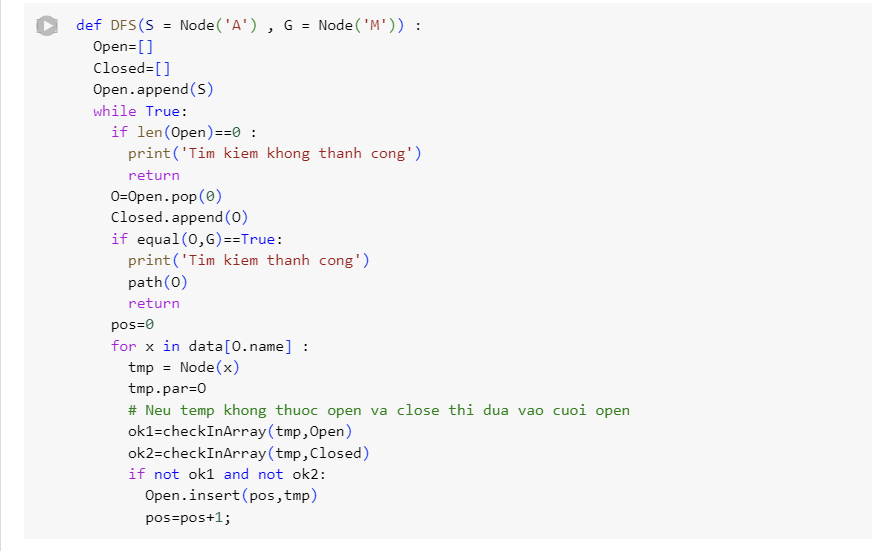
### **3.1.1 Thuật toán tìm kiếm chiều sâu**

**Mô tả thuật toán :**

****

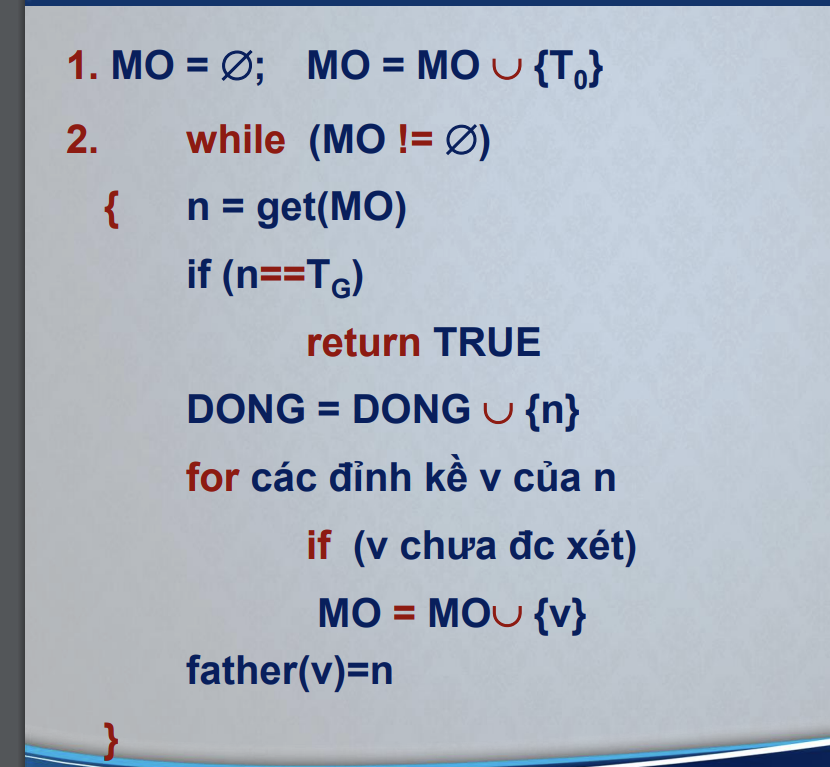
**Sử dụng ngôn ngữ python xây dựng thuật toán**

****

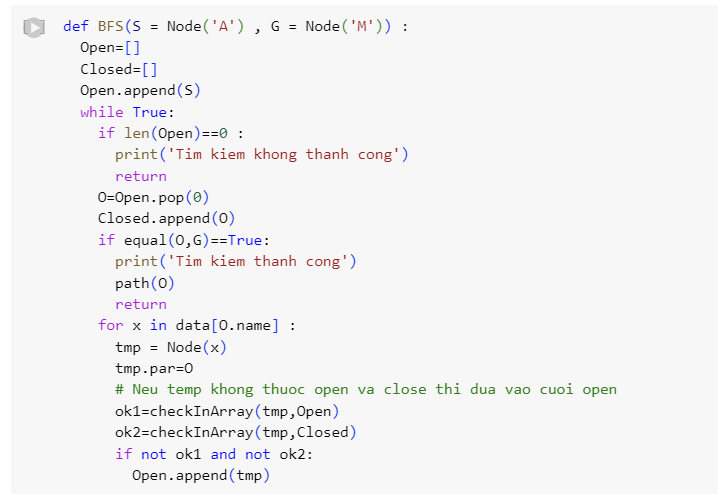
****

### **3.1.2 Thuật toán tìm kiếm chiều rộng**

**Mô tả thuật toán :**



**Sử dụng ngôn ngữ python xây dựng thuật toán**

****

## **3.2 Mô tả bài toán người lái đò**

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. Làm thế nào để có thể chở tất cả sang bên kia sông?

Các toán tử:

* Người từ bờ A sang bờ B
* Người từ bờ B quay lại bờ A
* Người và sói từ bờ A sang bờ B
* Người và sói từ bờ B quay lại bờ A
* Người và cải từ bờ A sang bờ B
* Người và cải từ bờ B sang bờ A
* Người và dê từ bờ A sang bờ B
* Người và dê từ bờ B sang bờ A

### **3.2.1 Không gian trạng thái**

Mô tả trạng thái theo thứ tự sau (Sói, Dê, Cải, Người)

Trạng thái bắt đầu: (0, 0, 0, 0) cả sói, dê, cải và người đang ở bờ A.

Trạng thái kết thúc: (1, 1, 1, 1) cả sói, dê, cải và người đã sang bờ B.

Mô tả không gian trạng thái của bài toán:

A diagram of a chemical structure

Description automatically generated

*Biều đồ 2. 1: Không gian trạng thái bài toán người lái đò*

### **3.2.2 Lời giải**

* Người từ bờ 1 sang bờ 2
* Người từ bờ 2 về bờ 1
* Người và Sói từ bờ 1 sang bờ 2
* Người và Sói từ bờ 2 về bờ 1
* Người và Dê bờ 1 sang bờ 2
* Người và Dê từ bờ 2 về bờ 1
* Người và Cải từ bờ 1 sang bờ 2
* Người và Cải từ bờ 2 về bờ 1

\*Chuyển đổi không gian trạng thái sang đồ thị:

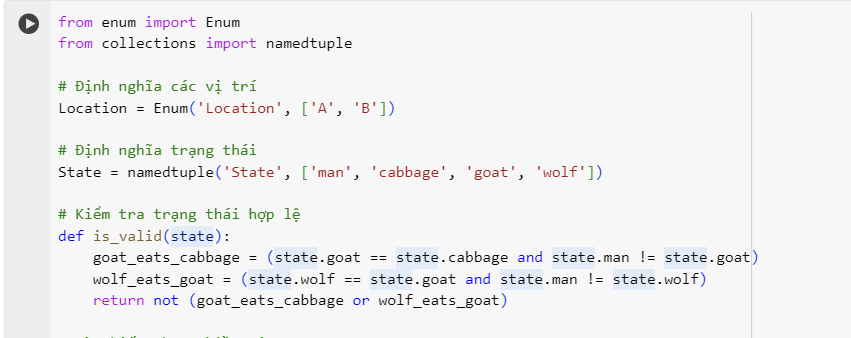
**A diagram of a molecule

Description automatically generated**

*Biều đồ 2. 2: Đồ thị bài toán người lái đò*

## **3.3 Cài đặt thuật toán**

Khởi tạo các biến cần thiết:



*Hình 2. 2: Khởi tạo hàm và thành phần cần thiết*

Xây dựng thuật toán tìm kiếm DFS:

def depth\_first\_search(start, is\_goal, get\_neighbors):

parent = dict() # tạo một từ điển để lưu trữ đỉnh cha của mỗi đỉnh

to\_visit = [start] # khởi tạo ngăn xếp stack

discovered = set([start]) # tạo một tập thể để lưu trữ các đỉnh đã được khám phá

while to\_visit:

vertex = to\_visit.pop() # lấy ra đỉnh của ngăn xếp

if is\_goal(vertex): # nếu là đỉnh đích

path = [] # khởi tạo một danh sách để lưu trữ đường đi

while vertex is not None:# cho qua các đỉnh

path.insert(0, vertex) # thêm đỉnh hiện tại vào danh sách đường đi

vertex = parent.get(vertex) # cập nhập đỉnh hiện tại thành đỉnh cha của nó

return path # trả về đường đi của đỉnh đỉnh bắt đầu đến đích

for neighbor in get\_neighbors(vertex): # lặp qua các đỉnh con đỉnh hiện tại

if neighbor not in discovered and is\_valid(neighbor): # nêu đỉnh chưa xét thì hợp lệ

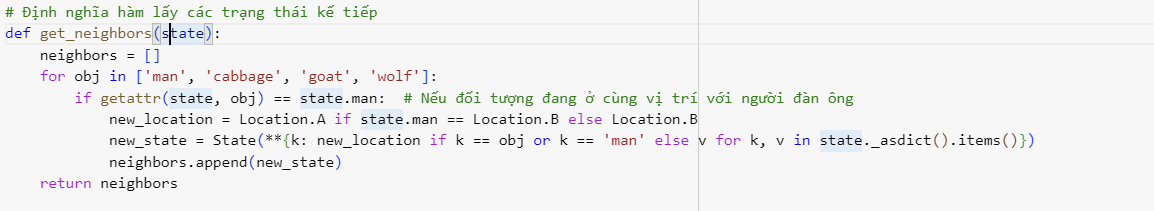
discovered.add(neighbor) # thêm đỉnh vào tập hợp đỉnh đã xét

parent[neighbor] = vertex # đặt hiện tại làm đỉnh cha

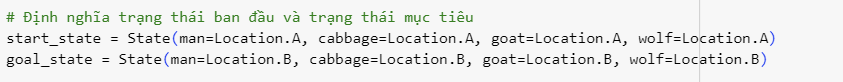
to\_visit.append(neighbor) # thêm đỉnh con vào ngăn xếp

*Hình 2. 3: Thuật toán DFS*

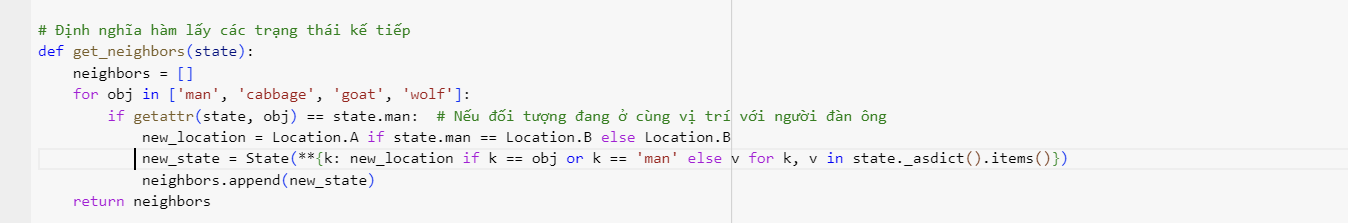
Hàm lấy các trạng thái kế tiếp :



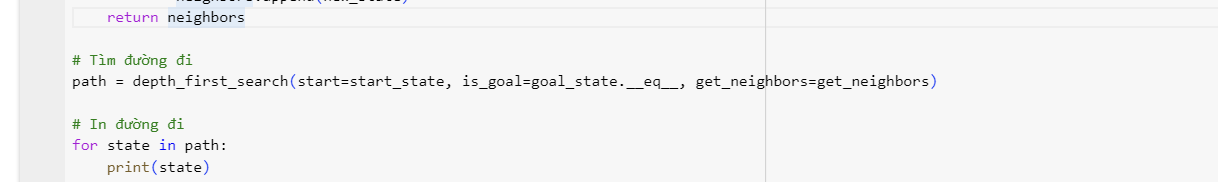
Khởi tạo trạng thái ban đầu :



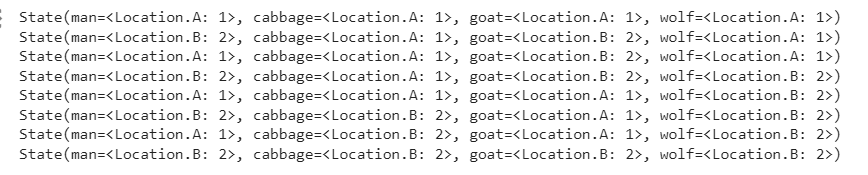
Hàm định nghĩa các trạng thái kế tiếp



Tìm và in ra đường đi :



Kết quả:



*Hình 2. 5: Chạy thử chương trình*

# 

# 

# 

# 

# **KẾT LUẬN**

Trong bài tập lớn lần này, chúng em đã tiến hành nghiên cứu và thực hiện ứng dụng thuật toán tìm kiếm mù vào bài toán người lái đò. Mục tiêu của bài tập này là tìm hiểu các phương pháp tìm kiếm mù, các phương pháp tìm kiếm heuristic và áp dụng các thuật toán tìm kiếm mù để vào bài toán người lái đò. Chúng em đã bắt đầu bằng xác định mục tiêu, phương hướng nghiên cứu cũng như phạm vi nghiên cứu, từ đó quy hoạch phạm vi kiến thức. Sau đó tìm hiểu các thuật toán tìm kiếm theo chiều sâu và chiều rộng, trong đó chú trọng đến việc xây dựng hàm và chiến lược di chuyển thông minh. Điều này đã giúp chúng em hiểu rõ hơn về không gian trạng thái, phương pháp tìm kiếm lời giải của bài toán người lái đò nói riêng và các bài toán áp dụng thuật toán tìm kiếm mù nói chung.

Qua báo cáo bài tập lớn này, nhóm chúng em đã hiểu rõ hơn về học phần trí tuệ nhân tạo nói chung và các thuật toán tìm kiếm mù, rút ra được nhiều kinh nghiệm qua bài tập thực hành. Mặt khác bài tập lớn này cũng giúp chúng em cải thiện nhiều kỹ năng mềm như việc khai thác thông tin từ Internet, tra cứu, kỹ năng làm việc nhóm và sử dụng các thông tin được viết bằng tiếng Anh.

Trong quá trình thực hiện bài tập lớn nhóm chúng em còn thiếu nhiều kiến thức về ngôn ngữ lập trình,… nên có thể có nhiều lỗi sai và thiếu sót mong cô bỏ qua cho. Nhóm chúng em cũng rất mong nhận được ý kiến đóng góp của cô để có thể học thêm được nhiều kinh nghiệm cho bài tập sau. Chúng em cũng xin chân thành cảm ơn sự hỗ trợ và hướng dẫn từ Ths. Nguyễn Lan Anh cũng như sự đóng góp của các thành viên trong nhóm trong suốt quá trình thực hiện dự án này.

# **TÀI LIỆU THAM KHẢO**

[1]. Giáo trình trí tuệ nhân tạo trường Đại học Công Nghiệp Hà Nội.

[2]. Nguồn internet