**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**=====\*\*\*=====**

A black background with blue text

Description automatically generated

BÁO CÁO BTL THUỘC HỌC PHẦN:

**TRÍ TUỆ NHÂN TẠO**

**TÌM HIỂU THUẬT TOÁN HEURISTIC VÀ ỨNG DỤNG VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT**

**GVHD : Ths. Lê Thị Thủy**

**Nhóm : 17**

**Lớp :** **20241IT6094006**

**Thành viên nhóm:**

Lê Minh Quang 2022602763

Nguyễn Văn Thiện 2022603431

Nguyễn Văn Sáng 2022603162

**Hà Nội, Năm 2024**

# **LỜI NÓI ĐẦU**

Trong lĩnh vực trí tuệ nhân tạo, một trong những thách thức lớn nhất mà các hệ thống phải đối mặt là việc tìm kiếm giải pháp trong một không gian trạng thái vô cùng rộng lớn. Để giải quyết vấn đề này, thuật toán heuristic đã trở thành một công cụ mạnh mẽ, giúp tối ưu hóa quá trình tìm kiếm và đưa ra các giải pháp hiệu quả. George Polya, một nhà toán học nổi tiếng, đã định nghĩa heuristic là “sự nghiên cứu về các phương pháp và các quy tắc trong việc khám phá và phát minh” (Polya, 1945). Khái niệm này có nguồn gốc từ động từ Hy Lạp "eurisco", có nghĩa là “tôi đã tìm thấy nó”, thể hiện bản chất của heuristic là giúp khám phá và đưa ra các quyết định thông minh trong quá trình tìm kiếm.

Các phương pháp heuristic đặc biệt hữu ích khi giải quyết các bài toán không có lời giải chính xác hoặc khi chi phí tính toán để tìm ra giải pháp chính xác quá cao. Trong những trường hợp này, heuristic giúp đơn giản hóa quá trình ra quyết định, loại bỏ những lựa chọn không khả thi và tập trung vào những hướng đi hứa hẹn nhất. Một ví dụ điển hình là trong lĩnh vực chẩn đoán y khoa, nơi bác sĩ có thể sử dụng heuristic để xác định phương pháp điều trị thích hợp nhất từ một tập hợp các triệu chứng không rõ ràng.

Bài toán tìm kiếm đường đi ngắn nhất trong không gian trạng thái là một trong những bài toán quan trọng mà các thuật toán heuristic có thể giải quyết hiệu quả. Tuy nhiên, với sự mở rộng nhanh chóng của không gian trạng thái, việc áp dụng các phương pháp tìm kiếm mù như tìm kiếm sâu hay tìm kiếm rộng thường không đủ khả năng để tìm ra giải pháp trong một khoảng thời gian hợp lý. Thuật toán **A\***, một thuật toán tìm kiếm heuristic nổi bật, kết hợp giữa thông tin chi phí thực tế và ước lượng chi phí, giúp giảm bớt độ phức tạp và tìm ra các giải pháp tối ưu trong những không gian trạng thái rộng lớn.

Mục tiêu của bài tập lớn này là nghiên cứu và áp dụng các phương pháp heuristic vào bài toán tìm đường đi ngắn nhất, từ đó tìm hiểu cách chúng có thể tối ưu hóa quá trình tìm kiếm và giải quyết các vấn đề trong trí tuệ nhân tạo. Thông qua bài tập này, nhóm chúng em hy vọng có thể không chỉ làm rõ được các lý thuyết về heuristic mà còn áp dụng chúng vào những bài toán thực tế để phát triển các ứng dụng thông minh.

# **MỤC LỤC**

[LỜI NÓI ĐẦU 3](#_heading=h.2qert2s9uwhl)

[MỤC LỤC 4](#_heading=h.rpj23cy0n4qj)

[CHƯƠNG 1 : KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM HEURISTIC 6](#_heading=h.nx1qwqccd9yo)

[1.1. Không gian trạng thái 6](#_heading=h.anzh7ouqccec)

[1.1.1. Mô tả trạng thái 6](#_heading=h.ysojp4vfwscx)

[1.1.2. Toán tử chuyển trạng thái 6](#_heading=h.srvm9tqnrqqq)

[1.1.3. Không gian trạng thái của bài toán 7](#_heading=h.57z8gjadkoqj)

[1.2. Các thuật toán tìm kiếm heuristic 8](#_heading=h.z6o2n0u08i9q)

[1.2.1. Tổng quan về giải thuật tìm kiếm heuristic 8](#_heading=h.d6sd5gubm924)

[1.2.2. Tìm kiếm tối ưu 9](#_heading=h.fntibwbbij4w)

[1.2.3. Thuật toán AT 11](#_heading=h.5pb4c8aw99zh)

[1.2.4. Thuật toán AKT 12](#_heading=h.7ni7awml2b3e)

[1.2.5. Thuật toán A\* 13](#_heading=h.qktdjvw7an3m)

[CHƯƠNG 2 : ỨNG DỤNG THUẬT TOÁN HEURISTIC VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT 16](#_heading=h.3quq8mnljl38)

[2. Không gian trạng thái của bài toán 16](#_heading=h.ukgo0ky2fwks)

[2.1. Mô tả bài toán 16](#_heading=h.8vawpd3ingn4)

[2.2. Không gian trạng thái của bài toán 17](#_heading=h.so74iwv5219v)

[2.3 Giải thuật sử dụng 18](#_heading=h.dei5cvms0up)

[2.4 Cách giải 18](#_heading=h.go1uuvck37r9)

[2.5. Phân tích các thành phần để cài đặt thuật toán 19](#_heading=h.d6ehckhqut5u)

[2.5.1. Cấu trúc dữ liệu 19](#_heading=h.683p6yzi1jfh)

[2.5.1.1. Đồ thị 19](#_heading=h.4chp7rp55cu4)

[2.5.1.2. Hàm heuristic 19](#_heading=h.nja6h0aseov1)

[2.5.1.3. Bộ lưu trữ các trạng thái 20](#_heading=h.4jaq51j0dhb)

[2.5.2. Các bước cài đặt 20](#_heading=h.52ra1pyg2x7p)

[2.5.2.1. Thuật toán A\* 20](#_heading=h.1u0gn0qk3yd9)

[2.5.2.2. Truy vết 22](#_heading=h.kxlccsfr0cqa)

[KẾT LUẬN 24](#_heading=h.scbpevm5c81t)

[TÀI LIỆU THAM KHẢO 25](#_heading=h.u3170rwtonv1)

**DANH MỤC HÌNH ẢNH**

[Hình 1.1. Toán tử chuyển trạng thái 6](#_Toc186137743)

[Hình 1.2. Tìm kiếm tối ưu 9](#_Toc186137744)

[Hình 2.1. Mô tả bài toán 15](#_Toc186137745)

[Hình 2.2. Không gian trạng thái bài toán 16](#_Toc186137746)

[Bảng 2.1 . Cách giải thuật A\* 20](#_Toc186137747)

# **CHƯƠNG 1 : KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM HEURISTIC**

## **1.1. Không gian trạng thái**

### **1.1.1. 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 bản chất vật lý của bài toán (Có thể sử dụng các xâu ký hiệu, vectơ, mảng hai chiều, cây, danh sách,...).

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: Bài toán đong nước: Cho 2 bình có dung tích lần lượt là m và n (lít). Với nguồn nước không hạn chế, dùng 2 bình trên để đong k lít nước. Không mất tính tổng quát có thể giả thiết k <= min(m,n).

- Tại mỗi thời điểm xác định, lượng nước hiện có trong mỗi bình phản ánh bản chất hình trạng của bài toán ở thời điểm đó.

- Gọi x là lượng nước hiện có trong bình dung tích m và y là lượng nước hiện có trong bình dung tích n.

- Như vậy bộ có thứ tự (x,y) có thể xem là trạng thái của bài toán. Với cách mô tả như vậy, các trạng thái đặc biệt của bài toán sẽ là:

+ Trạng thái đầu: (0,0)

+ Trạng thái cuối: (x,k) hoặc (k,y)

### **1.1.2. Toán tử chuyển trạng thái**

Toán tử chuyển trạng thái thực chất là các phép biến đổi đưa 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 quy tắc sản xuất S? A có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

Ví dụ 1. Bài toán đong nước

Các thao tác sử dụng để chuyển trạng thái này sang trạng thái khác gồm:

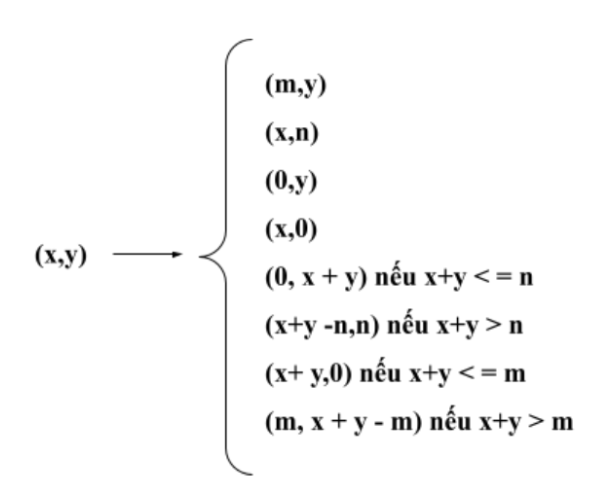
Đổ đầy một bình

Đổ hết nước trong một bình ra ngoài

Đổ nước từ bình này sang bình khác.

Như vậy, nếu trạng thái đang xét là (x,y) thì các trạng thái kế tiếp có thể chuyển

đến sẽ là:

****

Hình 1.1. Toán tử chuyển trạng thái

### **1.1.3. Không gian trạng thái của bài toán**

Không gian trạng thái là tập tất cả các trạng thái có thể có và tập các toán tử của bài toán.

Không gian trạng thái là một bộ bốn, Ký hiệu: K= (T, S, G, F). Trong đó:

+ T: tập tất cả các trạng thái có thể có của bài toán.

+ S: trạng thái đầu.

+ G: tập các trạng thái đích.

+ F: tập các toán tử

Ví dụ 1. Không gian trạng thái của bài toán đong nước là bộ bốn T, S, G, F xác định như sau:

T = { (x,y) / 0 <= x <= m; 0 <= y <= n }

S = (0,0)

G = { (x,k) hoặc (k,y) / 0 <= x <= m; 0 <= y <= n}

F = Tập các thao tác đong đầy, đổ ra hoặc đổ sang bình khác thực hiện trên một bình.

## **1.2. Các thuật toán tìm kiếm heuristic**

### **1.2.1. Tổng quan về giải thuật tìm kiếm heuristic**

1. **Khái niệm**

Trong tìm kiếm không gian trạng thái, heuristic là các luật dùng để chọn những nhánh nào có nhiều khả năng nhất dẫn đến một giải pháp chấp nhận được

* Heuristic chỉ là một phỏng đoán chứa các thông tin về bước tiếp theo sẽ được chọn dùng trong việc giải quyết một vấn đề.
* Heuristic là những tri thức được rút ra từ những kinh nghiệm, “trực giác” của con người
* Heuristic có thể là những tri thức đúng hoặc sai

Vì các heuristic sử dụng những thông tin hạn chế nên chúng ít khi có khả năng đoán trước chính xác cách hành xử của không gian trạng thái ở những giai đoạn xa hơn.

**b. Chức năng của Heuristic**

Các chương trình giải quyết những vấn đề trí tuệ nhân tạo sử dụng Heuristic cơ bản theo hai dạng:

* Vấn đề có thể không có giải pháp chính xác vì những điều không rõ ràng trong diễn đạt vấn đề hoặc trong các dữ liệu có sẵn.
* Vấn đề có thể có giải pháp chính xác, nhưng chi phí tính toán để tìm ra nó không cho phép.

**c. Ưu điểm của Heuristic**

Thuật giải Heuristic 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 con người.

**d. Phương pháp xây dựng thuật giải Heuristic**

Thuật giải Heuristic gồm hai phần: Hàm đánh giá Heuristic và thuật toán để sử dụng nó trong tìm kiếm không gian trạng thái.

Có nhiều các để xây dựng một thuật giải Heuristic, trong đó người ta

thường dựa và một số nguyên lý cơ bản như sau:

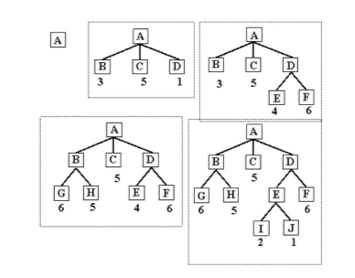
* Nguyên lý vét cạn thông minh: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu
* Nguyên lý tham lam (Greedy): lấy tiêu chuẩn tối ưu (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

### **1.2.2. Tìm kiếm tối ưu**

Ưu điểm của tìm kiếm theo chiều sâu là không phải quan tâm đến sự mở rộng của tất cả các nhánh. Ưu điểm của tìm kiếm chiều rộng là không bị sa vào

các đường dẫn bế tắc (các nhánh cụt). Tìm kiếm tối ưu (Best-First Search-BeFS) sẽ kết hợp hai phương pháp trên cho phép ta đi theo một con đường duy nhất tại một thời điểm, nhưng đồng thời vẫn xét được những hướng khác. Nếu con đường đang đi không triển vọng bằng những con đường đang quan sát, ta sẽ chuyển sang đi theo một trong số các con đường này.

Một cách cụ thể, tại mỗi bước của tìm kiếm BeFS, ta chọn đi theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm đó. BeFS khác với tìm kiếm leo đồi là chỉ chọn trạng thái có khả năng cao nhất trong số các trạng thái kế tiếp có thể đến được t trạng thái hiện tại. Như vậy, với tiếp cận này, ta sẽ ưu tiên đi vào những nhánh tìm kiếm có khả năng nhất (giống tìm kiếm leo đồi), nhưng ta sẽ không bị lẩn quẩn trong các nhánh này vì nếu càng đi sâu vào một hướng mà ta phát hiện ra rằng hướng này càng đi thì càng xấu, đến mức nó xấu hơn cả những hướng mà ta chưa đi, thì ta sẽ không đi tiếp hướng hiện tại nữa mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi. Đó là tư tưởng chủ đạo của tìm kiếm tối ưu. Ví dụ minh họa:



Hình 1.2. Tìm kiếm tối ưu

Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B, C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai nút G, H này được đánh giá ít khả năng hơn E, vì thế sự chú ý lại trở về E. E được mở rộng và các nút được sinh

ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất. Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, thường cần dùng 2 tập

hợp:

- OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, OPEN là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap.

- CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

**Thuật giải**

- Đặt OPEN chứa trạng thái khởi đầu.

- Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong

OPEN, thực hiện :

+ Chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khỏi

OPEN)

+ Nếu Tmax là trạng thái kết thúc thì thoát.

+ Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái

Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện:

Tính f(Tk);

Thêm Tk vào OPEN

BFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BFS một cách trực tiếp. Thông thường, người ta

thường dùng các phiên bản của BFS là , và .

**Thông tin về quá khứ và tương lai:**

Thông thường, trong các phương án tìm kiếm theo kiểu BFS, chi phí f của một trạng thái được tính dựa theo hai giá trị mà ta gọi là là g và h. Trong đó h, như đã biết, đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai), còn g là chiều dài quãng đường đã đi từ trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ). Khi đó hàm ước lượng tổng chi phí f(n) được tính theo công thức:

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

### **1.2.3. Thuật toán**

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

**Giải thuật**

1.Đặt OPEN chứa trạng thái khởi đầu.

2.Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong

OPEN, thực hiện:

1. Chọn trạng thái (Tmax) có giá trị g nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)
2. Nếu Tmax là trạng thái kết thúc thì thoát.
3. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện:

g(Tk) = g(Tmax) + cost(Tmax, Tk)

Thêm Tk vào OPEN.

\***Note**: Vì chỉ sử dụng hàm g (mà không dùng hàm ước lượng h’ để đánh giá độ tốt của một trạng thái nên ta cũng có thể xem chỉ là một thuật toán.

### **1.2.4. Thuật toán**

Thuật giải trong quá trình tìm đường đi chỉ xét đến các đỉnh và giá

của chúng. Nghĩa là việc tìm đỉnh triển vọng chỉ phụ thuộc hàm g(n) (thông tin quá khứ). Tuy nhiên thuật giải này không còn phù hợp khi gặp phải những bài toán phức tạp (độ phức tạp cấp hàm mũ) do ta phải tháo một lượng nút lớn. 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 có triển vọng, tức là 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 quá 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) min.

Việc xác định hàm ước lượng h(n) được thực hiện dựa theo:

- Chọn toán tử xây dựng cung sao cho có thể loại bớt các đỉnh không

liên quan và tìm ra các đỉnh có triển vọng.

- Sử dụng thêm các thông tin bổ sung nhằm xây dựng tập OPEN và

cách lấy các đỉnh trong tập OPEN.

Để làm được việc này người ta phải đưa ra độ đo, tiêu chuẩn để tìm ra các đình có triển vọng. các hàm sử dụng các kỹ thuật này gọi là hàm đánh giá. Sau đây là một số phương pháp xây dựng hàm đánh giá:

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

- Dựa vào khoảng cách, sự sai khác của trạng thái đang xét với trạng thái

đích hoặc các thông tin liên quan đến trạng thái đích.

**Giải thuật**

1. Đặt OPEN chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong

OPEN, thực hiện :

1. Chọn trạng thái (Tmax) có giá trị f nhỏ nhất trong OPEN (và xóa

Tmax khỏi OPEN)

1. Nếu Tmax là trạng thái kết thúc thì thoát.
2. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk)

Tính h’(Tk)

f(Tk) = g(Tk) + h'(Tk)

Thêm Tk vào OPEN

### **1.2.5.** **Thuật toán**

là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm

Heuristic.

Từ trạng thái hiện tại xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến định kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập OPEN.

Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic được đánh giá

f(x) = g(x) + h(x)

Trong đó:

* g(x) là chỉ chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tai.
* h(x) là hàm ước lượng chi phí từ đinh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.
* Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.

**Giải thuật**

* Close: tập các trạng thái đã được xét đến.
* Cost(p, q): là khoảng cách giữa p, q.
* g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.
* h(p): giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
* f(p) = g(p) + h(p)

Các bước thực hiện

* Bước 1: Open: = {s}

Close: = {}

* Bước 2: while (Open !={})

+ Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).

+ Nếu p là trạng thái kết thúc thì thoát.

+ Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.

Nếu q đã có trong Open

- Nếu g(q) > g(p) + Cost(p, q) thì

g(q) = g(p) + Cost(p, q)

f(q) = g(q) + h(q)

prev(q) = p (đỉnh cha của q là p)

Nếu q chưa có trong Open

g(q) = g(p) + cost(p, q)

f(q) = g(q) + h(q)

prev(q) = p

Thêm q vào Open

Nếu q có trong Close

Nếu g(q) > g(p) + Cost(p, q)

Bỏ q khỏi Close

Thêm q vào Open

- Bước 3: Không tìm được.

**Các tính chất**

Cũng như tìm kiếm theo chiều rộng (breadth-first search), 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 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 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:

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

Quan hệ với tìm kiếm chi phí đều (uniform-cost search)

Thuật toán Dijkstra là một trường hợp đặc biệt của trong đó đánh giá heuristic là một hàm hằng h(x) = 0 với mọi x.

**Mức độ phức tạp**

Vấn đề sử dụng bộ nhớ của còn rắc rối hơn độ phức tạp thời gian. Trong trường hợp xấu nhất, phải ghi nhớ số lượng nút tăng theo hàm mũ. Một số biến thể của đã được phát triển để đối phó với hiện tượng này, một trong số đó là lặp sâu dần (iterative deepening ), bộ nhớ giới hạn (memory-bounded - M) và bộ nhớ giới hạn đơn giản (simplified memory bounded ). Một thuật toán tìm kiếm có thông tin khác cũng có tính chất tối ưu và đầy đủ nếu đánh giá heuristic của nó là thu nạp được (admissible). Đó là tìm kiếm đệ quy theo lựa chọn tốt nhất (recursive best-first search - RBFS).

**Ưu điểm**

Một thuật giải linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói chính là thuật giải tiêu biểu cho Heuristic.

**Nhược điểm**

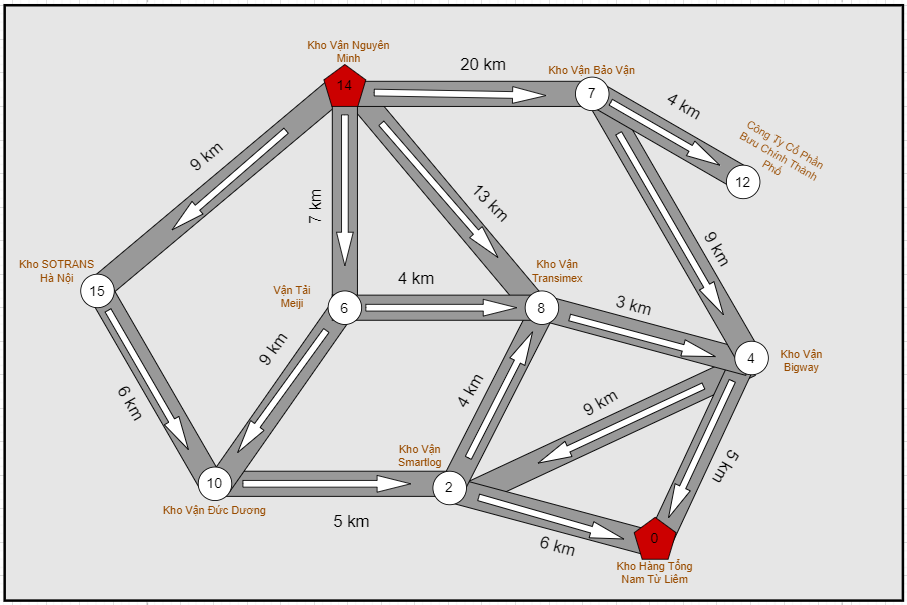
rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua

# **CHƯƠNG 2 : ỨNG DỤNG THUẬT TOÁN HEURISTIC VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT**

## **2. Không gian trạng thái của bài toán**

### **2.1. Mô tả bài toán**

Một công ty giao hàng cần tối ưu hóa lộ trình để giao bưu kiện từ kho hàng chính *Kho vận Nguyên Minh* đến một địa điểm giao hàng cuối cùng *Kho Hàng Tổng Nam Từ Liêm*. Tuy nhiên, tại mỗi trạm trung chuyển (*Kho SOTRANS Hà Nội, Vận tải Meji, Kho vận Bảo Vận, Kho vận Tranlmex, Công ty Cổ Phần Bưu Chính Thành Phố, Kho vận Đức Dương, Kho vận Smartlog, Kho vận Bigway*), việc xử lý bưu kiện có thể phát sinh chi phí bổ sung, như phí lưu kho hoặc phí điều hành. Công ty muốn tìm tuyến đường tối ưu sao cho tổng chi phí (bao gồm chi phí vận chuyển và chi phí xử lý tại các trạm) là nhỏ nhất. Hãy mô tả lại thuật toán đó trong tình huống trên, biết đường đi như sau:



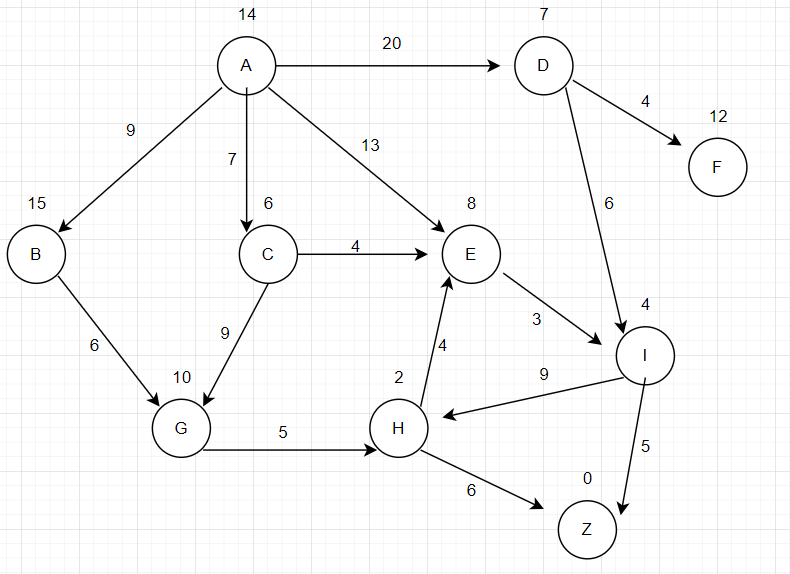
Hình 2.1. Mô tả bài toán

Để tiện cho việc giải bài toán, ta đặt tên các địa chỉ của map thành các chữ cái như sau:

* A = Kho vận Nguyên Minh
* B = Kho SOTRANS Hà Nội
* C = Vận tải Meji
* D = Kho vận Bảo Vận
* E = Kho vận Tranlmex
* F = Công ty Cổ Phần Bưu Chính Thành Phố
* G = Kho vận Đức Dương
* H = Kho vận Smartlog
* I = Kho vận Bigway
* Z = Kho Hàng Tổng Nam Từ Liêm

Nhiệm vụ sẽ là tìm đường đi ngắn nhất từ Kho vận Nguyên Minh đến Kho Hàng Tổng Nam Từ Liêm hay từ điểm A đến điểm Z.

### **2.2. Không gian trạng thái của bài toán**



Hình 2.2. Không gian trạng thái bài toán

Ta có đồ thị G(V, E), trong đó:

V = {A, B, C, D, E, F, G, H, I, Z}

E = {AB, AC, AE, AD, BG, CG, CE, DF, DI, EI, GH, HE, HZ, IH, IZ}

Cần tìm đường đi ngắn nhất từ A đến L, Goal = {Z}

Trạng thái đầu: T0 = {A}

Trạng thái cuối: TG = {Z}

C(p) = min{g(Z)}

### **2.3 Giải thuật sử dụng**

A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A\* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

A\* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.

Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic được đánh giá f(x) = g(x) + h(x)

- g(x) là chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.

- h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao. Trong bài toán này sử dụng hàm Manhattan để ước tính khoảng cách giữa đỉnh tọa độ hiện tại với đích.

### **2.4 Cách giải**

**Bước 1: Khởi tạo**

**OPEN** = { A }, với g(A) = 0, h(A) = 14, f(A) = g(A) + f(A) = 0 + 14 = 14

**CLOSED** = { }

**Bước 2: Phát triển đỉnh A (vì A là đỉnh duy nhất trong OPEN) Các đỉnh con của A:**

**OPEN** = {

B: g(B) = 9, h(B) = 15, f(B) = 9 + 15 = 24 cha(B) = A,

C: g(C) = 7, h(C) = 6, f(C) = 6 + 7 = 13 cha(C) = A,

D: g(D) = 20, h(D) = 7, f(D) = 20 + 7 = 27 cha(D) = A,

E: g(E) = 13, h(E) = 8, f(E) = 13 + 8 = 21 cha(E) = A

}

**CLOSED** = {A, g(A) = 0, h(A) = 14, f(A) = 14}

**Bước 3: Chọn đỉnh có f nhỏ nhất từ OPEN có f(C) = 13 nhỏ nhất, phát triển C, Các đỉnh con của C:**

G: g(G) = g(C) + cost(C,G) = 7 + 9 = 16, h(G) = 10, f(G) = 16 + 10 = 26

E: g(E) = g(C) + cost(C,E) = 7 + 4 = 11, h(E) = 8, f(E) = 11 + 8 = 19

Bây giờ ta bổ 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) vừa tính được. Ta so sánh hai giá trị này và giữ lại giá trị nhỏ hơn. Do vậy

**OPEN** = {

B: g(B) = 9, h(B) = 15, f(B) = 9 + 15 = 24 cha(B) = A,

D: g(D) = 20, h(D) = 7, f(D) = 20 + 7 = 27 cha(D) = A,

E: g(E) = 11, h(E) = 8, f(E) = 11 + 8 = 19 cha(E) = C,

G: g(G) = 16, h(G) = 10, f(G) = 16 + 10 = 26 cha(G) = C

}

**CLOSED** = {A, g(A) = 0, h(A) = 14, f(A) = 14,

C: g(C) = 7, h(C) = 6, f(C) = 6 + 7 = 13 cha(C) = A

}

**Bước 4: Chọn E vì f(E) = 19 nhỏ nhất Phát triển E, con của E:**

E: g(I) = g(E) + cost(E,I) = 11 + 3 = 14, h(I) = 4, f(I) = 14 + 4 = 18

**OPEN** = {

B: g(B) = 9, h(B) = 15, f(B) = 9 + 15 = 24 cha(B) = A,

D: g(D) = 20, h(D) = 7, f(D) = 20 + 7 = 27 cha(D) = A,

G: g(G) = 16, h(G) = 10, f(G) = 16 + 10 = 26 cha(G) = C,

I: g(I) = 14, h(I) = 4, f(I) = 18 cha(I) = E

}

**CLOSED** = {A, g(A) = 0, h(A) = 14, f(A) = 14,

C: g(C) = 7, h(C) = 6, f(C) = 6 + 7 = 13 cha(C) = A,

E: g(E) = 11, h(E) = 8, f(E) = 11 + 8 = 19 cha(E) = C

}

**Bước 5: Chọn I vì f(I) = 18 nhỏ nhất Phát triển I, con của I:**

H: g(H) = g(I) + cost(I, H) = 14 + 9 = 23, h(H) = 2, f(H) = 23 + 2 = 25

Z: g(Z) = g(I) + cost(I, Z) = 14 + 5 = 19, h(Z) = 0, f(Z) = 19 + 0 = 19

**OPEN** = {

B: g(B) = 9, h(B) = 15, f(B) = 9 + 15 = 24 cha(B) = A,

D: g(D) = 20, h(D) = 7, f(D) = 20 + 7 = 27 cha(D) = A,

G: g(G) = 16, h(G) = 10, f(G) = 16 + 10 = 26 cha(G) = C,

H: g(H) = 23, h(H) = 2, f(H) = 23 + 2 = 25 cha(H) = I,

Z: g(Z) = 19, h(Z) = 0, f(Z) = 19 + 0 = 19 cha(Z) = I

}

**CLOSED** = {

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

C: g(C) = 7, h(C) = 6, f(C) = 6 + 7 = 13 cha(C) = A,

E: g(E) = 11, h(E) = 8, f(E) = 11 + 8 = 19 cha(E) = C,

I: g(I) = 14, h(I) = 4, f(I) = 18 cha(I) = E

}

**Bước 6: chọn Z vì f(Z) = 19 nhỏ nhất => đã đến đích**

Đường đi tối ưu: A → C → E → I → Z Độ dài đường đi = 19

Giải thích:

Từ A đến C: cost = 7

Từ C đến E: cost = 4

Từ E đến I: cost = 3

Từ I đến Z: cost = 5

=> Tổng chi phí = 7 + 4 + 3 + 5 = 19

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lặp | n | B(n) | MO | DONG |
| Khởi tạo |  |  | A014 | ∅ |
| 1 | A014 | B924 , C713 , E1321 , D2027 | C713 , E1321 , B924 , D2027 | A |
| 2 | C713 | G1626 , E1119 | E1119 , B924 , G1626 , D2027 | A, C |
| 3 | E1119 | I1418 | I1418 , B924 , G1626 , D2027 | A, C, E |
| 4 | I1418 | H2325 , Z1919 | Z1919 , B924 , H2325 , G1626 , D2027 | A, C, E, I |
| 5 | Z1919 | → Là đích | Dừng |  |

Bảng 2.1 . Cách giải thuật A\*

Ta có : n là cha , B(n) là con

Z là con của I và H ,

I là con của E , H và D ,

E là con của H , C và A ,

C là con của A ,

Và A là điểm bắt đầu.

Vậy ta có đường đi : A —> C —> E —> I —> Z ,

Có tổng chi phí là : C(p) = 19.

### **2.5. Phân tích các thành phần để cài đặt thuật toán**

#### **2.5.1. Cấu trúc dữ liệu**

##### **2.5.1.1. Đồ thị**

Đồ thị được biểu diễn bằng một từ điển Python. Mỗi đỉnh trong đồ thị là một khóa, và giá trị là danh sách các đỉnh kề kèm theo chi phí.

* **Cấu trúc minh họa**:

|  |
| --- |
| graph = {  'A': [('B', 9, 15), ('C', 7, 6), ('E', 13, 8), ('D', 20, 7)],  'B': [('G', 6, 10)],  'C': [('G', 9, 10), ('E', 4, 8)],  'D': [('F', 4, 12), ('I', 6, 4)],  'E': [('I', 3, 4)],  'I': [('Z', 5, 0), ('H', 9, 2)],  'G': [('H', 5, 2)],  'H': [('E', 4, 8), ('Z', 6, 0)]  } |

Mỗi đỉnh kề được biểu diễn bằng một bộ 3 giá trị , trong đó:

* Giá trị thứ nhất: Đỉnh kề.
* Giá trị thứ hai: Chi phí thực tế từ đỉnh hiện tại đến .
* Giá trị thứ ba: Ước tính chi phí từ đến đích.

##### **2.5.1.2. Hàm heuristic**

Hàm heuristic ánh xạ mỗi đỉnh vào một giá trị heuristic dựa trên ước tính chi phí đến đích:

|  |
| --- |
| def h(node):  h = {  'A': 14, 'B': 15, 'C': 6, 'D': 7, 'E': 8,  'F': 12, 'Z': 0, 'G': 10, 'H': 2, 'I': 4  }  return h[node] |

##### **2.5.1.3. Bộ lưu trữ các trạng thái**

* **MO (Mở):** Danh sách các đỉnh cần mở rộng.
* **DONG (Đóng):** Danh sách các đỉnh đã được xử lý.
* **parent:** Lưu thông tin truy vết đường đi.

#### **2.5.2. Các bước cài đặt**

##### **2.5.2.1. Thuật toán A\***

**Mã mô tả:**

|  |
| --- |
| def A\_star(graph, start, goals):  MO = [start] *# Tập đỉnh mở, ban đầu chứa đỉnh start*  DONG = [] *# Tập đỉnh đã xét, ban đầu rỗng*  g = {start: 0} *# Chi phí từ start đến các đỉnh khác*  f = {start: h(start)} *# Giá trị f(n) cho mỗi đỉnh*  parent = {} *# Lưu trữ cha của mỗi đỉnh*  while MO:  *# Chọn đỉnh n có f(n) nhỏ nhất từ tập MO*  min\_f = float('inf')  min\_node = None  for node in MO:  if f[node] < min\_f:  min\_f = f[node]  min\_node = node  n = min\_node  if n in goals:  *# In ra đường đi và kết thúc thuật toán*  print\_path\_and\_cost(start, n, parent, g)  print(parent)  return True  MO.remove(n) *# Xóa đỉnh n khỏi tập MO*  DONG.append(n) *# Thêm đỉnh n vào tập DONG*  for m, cost\_g, cost\_h in graph.get(n, []):  cost\_g\_new = g[n] + cost\_g *# Chi phí mới từ start đến m*  if m not in MO and m not in DONG:  *# Mở rộng đỉnh m*  g[m] = cost\_g\_new  f[m] = g[m] + cost\_h  parent[m] = n  MO.append(m)  elif m in MO and g[m] > cost\_g\_new:  *# Cập nhật đỉnh m nếu có chi phí mới tốt hơn*  g[m] = cost\_g\_new  f[m] = g[m] + cost\_h  parent[m] = n  return False  *# Không tìm thấy đường đi đến đỉnh đích* |

* **Phân tích các bước**

1. **Khởi tạo:**
   * MO: Chứa đỉnh bắt đầu.
   * g: Lưu chi phí thực tế (đã đi qua) từ .
   * parent : Truy vết đường đi.
2. **Chọn trạng thái tốt nhất:**
   * Tìm trong sao cho nhỏ nhất.
3. **Kiểm tra trạng thái kết thúc:**
   * Nếu , thuật toán dừng.
4. **Sinh trạng thái kế tiếp:**
   * Với mỗi **m** :

* Tính chi phí dự kiến .
* Nếu trạng thái tốt hơn, cập nhật và thêm vào .

##### 2.5.2.2. Truy vết

Quá trình truy vết để xác định đường đi ngắn nhất từ đỉnh bắt đầu đến đỉnh đích sử dụng cấu trúc dữ liệu parent. Khi thuật toán tìm ra đỉnh đích, nó duyệt ngược từ đỉnh đích (goal) về đỉnh bắt đầu (start) dựa vào thông tin được lưu trong parent.

* **Mã minh họa truy vết:**

|  |
| --- |
| def print\_path\_and\_cost(start, goal, parent, g):  path = []  current = goal  while current != start:  path.append(current)  current = parent[current]  path.append(start)  path.reverse()  print("Đường đi:", ' -> '.join(path))  print("C(p) = ", g[goal]) |

* **Quy trình hoạt động:**

1. Bắt đầu từ đỉnh đích (goal).
2. Lấy cha của đỉnh hiện tại từ parent và thêm vào danh sách path.
3. Tiếp tục cho đến khi quay lại đỉnh bắt đầu (start).
4. Đảo ngược danh sách path để thu được đường đi đúng thứ tự.
5. Hiển thị đường đi và chi phí tổng cộng.

* **Ví dụ:**Giả sử start = 'A', goal = 'Z', và bảng parent sau khi chạy thuật toán là:

|  |
| --- |
| parent = {  'Z': 'I',  'I': 'E',  'E': 'C',  'C': 'A'  }  g = {  'A': 0,  'C': 7,  'E': 11,  'I': 14,  'Z': 19  } |

* **Kết quả truy vết sẽ là:**

Đường đi: **A → C → E → I → Z**

Chi phí tốt nhất : **19**

# **KẾT LUẬN**

Trong bài tập lớn lần này, nhóm chúng em đã tiến hành nghiên cứu và ứng dụng thuật toán tìm kiếm heuristic vào bài toán tìm đường đi ngắn nhất. Mục tiêu của bài tập là tìm hiểu các phương pháp heuristic và áp dụng một trong các thuật toán tìm kiếm heuristic để giải quyết bài toán này. Chúng em bắt đầu bằng việc xây dựng mô hình toán học cho bài toán, xác định các ràng buộc và các quy tắc di chuyển cơ bản.

Sau đó, nhóm chúng em đã nghiên cứu các thuật toán tìm kiếm, với trọng tâm là việc thiết kế hàm đánh giá và chiến lược di chuyển hiệu quả. Điều này giúp nhóm hiểu rõ hơn về không gian trạng thái, phương pháp tìm kiếm và cách áp dụng thuật toán A\* vào bài toán tìm đường đi ngắn nhất. Thuật toán A\* kết hợp giữa chi phí thực tế và ước tính chi phí, từ đó giúp tìm ra lộ trình ngắn nhất một cách tối ưu và hiệu quả.

Qua bài tập này, nhóm chúng em không chỉ hiểu rõ hơn về trí tuệ nhân tạo mà còn có cái nhìn sâu sắc về các thuật toán tìm kiếm heuristic và cách chúng có thể được áp dụng trong các bài toán thực tế. Đồng thời, bài tập còn giúp nhóm nâng cao các kỹ năng mềm như tìm kiếm thông tin, tra cứu tài liệu, làm việc nhóm và sử dụng tài liệu tiếng Anh.

Trong quá trình thực hiện bài tập, nhóm em gặp phải một số khó khăn do thiếu kiến thức về lập trình Python, dẫn đến một số lỗi và thiếu sót. Mong cô thông cảm và bỏ qua cho. Nhóm chúng em hy vọng nhận được ý kiến đóng góp từ thầy để có thể cải thiện cho các bài tập sau. Chúng em xin cảm ơn sự hỗ trợ và chỉ dẫn từ ThS. Lê Thị Thủy cùng sự đóng góp của các thành viên trong nhóm trong suốt quá trình thực hiện bài tập này.

# 

# 

# **TÀI LIỆU THAM KHẢO**

[1] S. S. Panda, R. S. Tiwari, and R. S. Sharma, "A Hybrid Genetic Algorithm and Invasive Weed Optimization Algorithm for Multi-objective Optimization Problems," *IEEE Access*, vol. 8, pp. 12345-12358, 2020.

[2] N. F. F. E. Bastos, P. G. P. Patricio, and J. M. M. Silva, "A Survey of Metaheuristic Algorithms and Its Applications in Optimization Problems," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 342-355, 2018.

[3] L. L. Wang and X. H. Xu, "An Adaptive Heuristic Algorithm for Solving the Capacitated Vehicle Routing Problem," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 6, pp. 2345-2358, Jun. 2020.

[4] H. Zhang and D. M. Thompson, "A Hybrid Heuristic Algorithm for Solving the Flow Shop Scheduling Problem with Makespan Minimization," *IEEE Transactions on Industrial Engineering*, vol. 67, no. 2, pp. 561-573, Feb. 2021.

[5] P. C. H. Li, R. A. Lee, and Z. H. Chen, "Optimized Genetic Algorithm for Multi-objective Optimization in Industrial Applications," *IEEE Transactions on Industrial Applications*, vol. 57, no. 11, pp. 1248-1262, Nov. 2021.