**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

…□□□…

# BÁO CÁO BTL THUỘC HỌC PHẦN:
# TRÍ TUỆ NHÂN TẠO

## TÌM HIỂU THUẬT TOÁN BREADTH-FIRST SEARCH VÀ ÁP DỤNG VÀO BÀI TOÁN ĐONG NƯỚC

**GVHD:** Ths.Trần Thanh Huân

**Nhóm - Lớp:** 07 – 20242IT6094001

| | | |
|---|---|---|
| **Thành viên:** | Vũ Thị Phượng | MSV: 2023605711 |
| | Đào Duy Sơn | MSV: 2023606919 |
| | Nguyễn Hà Nam | MSV: 2023606322 |
| | Đặng Thái Sơn | MSV: 2023604646 |

***Hà Nội, năm 2025***

# LỜI CẢM ƠN

Trước hết, nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến giảng viên hướng dẫn**:** Trần Thanh Huân – người đã tận tình chỉ dẫn, góp ý và tạo điều kiện thuận lợi để nhóm hoàn thành đề tài *"Tìm hiểu thuật toán Breadth-First Search và áp dụng vào bài toán đong nước*".

Chúng em cũng xin trân trọng cảm ơn khoa và nhà trường đã cung cấp môi trường học tập thuận lợi cùng các tài liệu học thuật quý báu, giúp nhóm có cơ hội tiếp cận và thực hiện đề tài này một cách nghiêm túc và hiệu quả.

Bên cạnh đó, nhóm cũng xin cảm ơn các thành viên trong nhóm đã cùng nhau làm việc tích cực, chia sẻ ý tưởng, phân công công việc hợp lý và phối hợp nhịp nhàng trong suốt quá trình thực hiện nghiên cứu.

Mặc dù nhóm đã cố gắng hết sức để hoàn thiện đề tài, tuy nhiên do giới hạn về thời gian và kinh nghiệm, bài nghiên cứu khó tránh khỏi những thiếu sót nhất định. Nhóm rất mong nhận được sự thông cảm và những góp ý quý báu từ thầy cô và bạn bè để có thể cải thiện hơn trong các đề tài nghiên cứu sau. Nhóm xin chân thành cảm ơn!

**Nhóm 7 thực hiện**

# MỤC LỤC

# MỞ ĐẦU

## 1. Lý do chọn đề tài

Trong thời đại công nghệ số phát triển mạnh mẽ, Trí tuệ nhân tạo (Artificial Intelligence – AI) ngày càng khẳng định vị thế quan trọng trong nhiều lĩnh vực như y tế, giáo dục, công nghiệp, giao thông, và các hệ thống hỗ trợ ra quyết định. Một trong những thành phần cốt lõi của AI chính là các thuật toán tìm kiếm, đóng vai trò quan trọng trong việc xác định chiến lược hành động, lập kế hoạch và tối ưu hóa quá trình ra quyết định của máy.

Trong đó, thuật toán Breadth-First Search (BFS) là một thuật toán kinh điển thuộc nhóm tìm kiếm có hệ thống, được ứng dụng phổ biến trong các bài toán cần tìm lời giải ngắn nhất trong không gian trạng thái rời rạc. Tuy đơn giản về mặt cấu trúc, BFS lại rất hiệu quả trong việc đảm bảo tìm ra lời giải tối ưu nếu có, nên thường được dùng để giải quyết nhiều bài toán thực tiễn cũng như phục vụ cho mục đích nghiên cứu.

Bài toán đong nước là một ví dụ tiêu biểu trong nhóm bài toán tìm kiếm trong không gian trạng thái. Dù bài toán chỉ xoay quanh hai chiếc bình và các thao tác đổ, rót, nhưng lại đòi hỏi người giải phải có tư duy logic, khả năng mô hình hóa trạng thái, xác định các thao tác hợp lệ và lựa chọn chiến lược duyệt thông minh để tìm ra lời giải hiệu quả.

Xuất phát từ nhu cầu tìm hiểu sâu hơn về các thuật toán tìm kiếm trong trí tuệ nhân tạo, cũng như mong muốn vận dụng lý thuyết vào giải quyết một bài toán cụ thể, nhóm đã lựa chọn đề tài "Tìm hiểu thuật toán Breadth-First Search và áp dụng vào bài toán đong nước" làm hướng nghiên cứu và thực hành cho học phần Trí tuệ nhân tạo.

Đề tài không chỉ góp phần củng cố kiến thức về thuật toán BFS mà còn giúp người học rèn luyện khả năng lập trình, tư duy phân tích và thiết kế giải pháp cho các bài toán mang tính mô hình hóa trong thực tế.

## 2. Mục tiêu nghiên cứu

Mục tiêu chính của đề tài là nghiên cứu nguyên lý hoạt động và khả năng ứng dụng của thuật toán Breadth-First Search (BFS) trong việc giải quyết một bài toán cụ thể thuộc nhóm bài toán không gian trạng thái – bài toán đong nước.

Thông qua việc cài đặt và mô phỏng thuật toán BFS, đề tài hướng đến:

- Làm rõ cách thức thuật toán BFS vận hành trong môi trường tìm kiếm rời rạc, nơi các trạng thái được xác định và chuyển tiếp thông qua các thao tác hợp lệ.
- Hiểu rõ bản chất chiến lược duyệt theo chiều rộng, từ đó nhận diện ưu điểm của BFS trong việc tìm ra lời giải ngắn nhất (tối ưu về số bước).
- Ứng dụng lý thuyết vào thực tiễn, bằng cách mô hình hóa bài toán đong nước thành không gian trạng thái và áp dụng thuật toán để tìm ra chuỗi thao tác dẫn đến trạng thái mục tiêu.
- Nâng cao khả năng tư duy giải thuật và lập trình giải quyết vấn đề, từ bước phân tích bài toán đến thiết kế giải pháp và đánh giá hiệu quả.

## 3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài là thuật toán tìm kiếm Breadth-First Search (BFS) và cách áp dụng nó để giải quyết bài toán Đong nước trong trí tuệ nhân tạo. Đề tài tập trung làm rõ nguyên lý hoạt động của BFS và quá trình xây dựng không gian trạng thái cho bài toán cụ thể.

Phạm vi nghiên cứu bao gồm:

- Mô hình hóa bài toán đong nước dưới dạng không gian trạng thái gồm các cặp (x,y) đại diện cho lượng nước hiện có trong hai bình.
- Xây dựng thuật toán tìm kiếm theo chiều rộng (BFS) để duyệt qua các trạng thái và tìm ra lời giải hợp lệ với số bước ít nhất.

- Lập trình thuật toán bằng ngôn ngữ Python, sử dụng các cấu trúc dữ liệu như hàng đợi (deque) và tập hợp (set) để tối ưu hóa việc tìm kiếm.
- Đánh giá hiệu quả giải thuật thông qua thời gian thực thi và số bước thực hiện trên nhiều trường hợp mục tiêu khác nhau.
- Hạn chế phạm vi trong các thao tác cơ bản như: đổ đầy bình, rót nước từ bình này sang bình kia, hoặc đổ bỏ nước – không mở rộng sang các yếu tố thực tế như giới hạn thời gian, thất thoát nước hay sai số đo lường.

## 4. Kết quả mong muốn

Thông qua quá trình nghiên cứu và thực nghiệm, nhóm mong muốn đạt được các kết quả sau:

- Củng cố kiến thức lý thuyết đặc biệt là các thuật toán tìm kiếm trong lĩnh vực trí tuệ nhân tạo.
- Hiểu rõ nguyên lý hoạt động của thuật toán tìm kiếm BFS.
- Xây dựng được mô hình không gian trạng thái phù hợp cho bài toán đong nước.
- Cài đặt thành công chương trình giải bài toán đong nước bằng thuật toán BFS.
- Phân tích, đánh giá hiệu quả thuật toán khi áp dụng vào bài toán cụ thể.

## 5. Cấu trúc của báo cáo

Báo cáo gồm các phần chính sau:

- **Chương I:** Tổng quan về trí tuệ nhân tạo và tìm kiếm trong AI
- **Chương II:** Thuật toán Breadth-First Search và áp dụng vào bài toán đong nước
- **Chương III:** Cài đặt và thực nghiệm

- **Kết luận:** Tổng kết nội dung nghiên cứu, đánh giá kết quả đạt được và định hướng phát triển trong tương lai.

# CHƯƠNG I: TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO VÀ TÌM KIẾM TRONG AI

## 1.1. Giới thiệu về trí tuệ nhân tạo

Trí tuệ nhân tạo (AI) là khả năng giúp các hệ thống máy tính thực hiện những nhiệm vụ vốn đòi hỏi trí thông minh của con người, chẳng hạn như học tập, suy luận, giải quyết vấn đề, nhận thức môi trường và ra quyết định. Đây là một lĩnh vực nghiên cứu thuộc khoa học máy tính, tập trung vào việc phát triển các phương pháp và công nghệ nhằm giúp máy móc có thể nhận thức, học hỏi và hành động một cách thông minh.

Trong thực tế, AI đã được ứng dụng rộng rãi trong nhiều lĩnh vực của đời sống. Các ví dụ nổi bật bao gồm: công cụ tìm kiếm thông minh (Google Search), hệ thống gợi ý sản phẩm (YouTube, Amazon, Netflix), trợ lý ảo (Google Assistant, Siri, Alexa), xe tự lái (Waymo), các công cụ tạo nội dung (ChatGPT, AI art) và các hệ thống có khả năng chơi game chiến lược ở trình độ cao (như cờ vua, cờ vây). Thậm chí, nhiều công nghệ AI hiện đại đã trở thành một phần quen thuộc trong các ứng dụng hàng ngày và không còn được người dùng nhận diện rõ là AI.

AI gồm nhiều phân ngành nghiên cứu với các mục tiêu cụ thể như học máy, lập luận, biểu diễn tri thức, lập kế hoạch, xử lý ngôn ngữ tự nhiên, thị giác máy tính và điều khiển robot. Để đạt được các mục tiêu này, các nhà nghiên cứu kết hợp nhiều kỹ thuật khác nhau, từ tối ưu hóa toán học, logic hình thức, mạng nơ-ron nhân tạo cho đến các phương pháp thống kê. Ngoài ra, AI còn kế thừa nhiều kiến thức từ các ngành như tâm lý học, ngôn ngữ học, triết học và khoa học thần kinh.

Hiện nay, các công ty công nghệ hàng đầu như OpenAI, Google DeepMind và Meta đang nỗ lực phát triển Trí tuệ nhân tạo tổng quát (AGI) — một loại AI có thể thực hiện hầu hết các nhiệm vụ trí tuệ với hiệu suất ngang bằng hoặc cao hơn con người.

Lịch sử của AI đã trải qua nhiều giai đoạn phát triển thăng trầm. Được hình thành từ năm 1956, lĩnh vực này từng đối mặt với các thời kỳ "mùa đông AI" do sự kỳ vọng không được đáp ứng và nguồn tài trợ bị cắt giảm. Tuy nhiên, từ năm 2012, với sự ra đời của các mạng nơ-ron sâu (deep learning) và khả năng xử lý mạnh mẽ của GPU, AI đã có những bước tiến vượt bậc. Đặc biệt, từ năm 2017 với sự phát triển của kiến trúc Transformer và trong những năm gần đây, các công nghệ AI tạo sinh (generative AI) đã mở ra một làn sóng ứng dụng mới. Sự bùng nổ của AI hiện đại cũng đồng thời đặt ra nhiều thách thức và câu hỏi đạo đức, buộc các nhà nghiên cứu và nhà hoạch định chính sách phải cân nhắc kỹ lưỡng về quản lý, tính an toàn và lợi ích lâu dài của AI đối với xã hội.

### 1.2. Phân loại bài toán và vai trò của tìm kiếm

Trong lĩnh vực trí tuệ nhân tạo nói riêng và khoa học máy tính nói chung, bài toán tìm kiếm đóng vai trò nền tảng trong việc giải quyết nhiều dạng bài toán khác nhau. Nhìn chung, các bài toán trong AI có thể được phân loại thành hai nhóm chính:

- **Bài toán có không gian trạng thái hữu hạn và xác định:** các bài toán mà không gian trạng thái có thể được mô tả một cách rõ ràng và có thể liệt kê hết, ví dụ như bài toán đong nước, bài toán 8 quân hậu, bài toán tìm đường trên bản đồ...
- **Bài toán có không gian trạng thái liên tục hoặc không xác định:** các bài toán phức tạp hơn như xử lý ngôn ngữ tự nhiên, nhận diện hình ảnh, hoặc điều khiển robot trong môi trường thực, nơi không gian trạng thái rất lớn hoặc không thể mô tả một cách tường minh.

Trong cả hai nhóm bài toán này, vai trò của tìm kiếm là vô cùng quan trọng. Thuật toán tìm kiếm giúp hệ thống có khả năng duyệt qua các trạng thái tiềm năng, đánh giá và lựa chọn các bước đi phù hợp để tiến gần hơn đến mục tiêu.

Đối với các bài toán có không gian trạng thái hữu hạn và xác định, thuật toán tìm kiếm bảo đảm tìm được lời giải nếu có hoặc thông báo không có lời

giải nếu không tồn tại. Các thuật toán phổ biến như BFS (Breadth-First Search), DFS (Depth-First Search), hay các thuật toán tìm kiếm có trọng số như A* thường được áp dụng hiệu quả cho nhóm bài toán này.

Ngoài ra, trong các bài toán phức tạp hơn, thuật toán tìm kiếm kết hợp với các kỹ thuật học máy, heuristic hoặc các chiến lược tối ưu hóa sẽ giúp hệ thống đưa ra các hành động hợp lý trong không gian rất lớn hoặc không đầy đủ thông tin.

Như vậy, tìm kiếm không chỉ là công cụ cốt lõi trong việc giải quyết bài toán mà còn là thành phần quan trọng giúp các hệ thống AI có khả năng ra quyết định thông minh và hiệu quả.

### 1.3. Thuật toán tìm kiếm trong không gian trạng thái

Trong lĩnh vực Trí tuệ nhân tạo (AI), một trong những cách tiếp cận phổ biến để giải quyết bài toán là mô hình hóa nó dưới dạng không gian trạng thái. Không gian trạng thái là tập hợp tất cả các trạng thái mà hệ thống có thể đạt được, cùng với các phép toán (hoặc hành động) cho phép chuyển đổi từ trạng thái này sang trạng thái khác.

Để mô tả một bài toán tìm kiếm trong không gian trạng thái, ta cần xác định các thành phần cơ bản sau:

- Trạng thái khởi đầu (Initial State): là điểm xuất phát của quá trình tìm kiếm.
- Trạng thái mục tiêu (Goal State): là đích đến mà thuật toán hướng tới.
- Tập hợp trạng thái (State Space): gồm toàn bộ các trạng thái có thể xảy ra trong bài toán.
- Tập các phép toán chuyển trạng thái (Operators): mô tả cách di chuyển từ trạng thái này sang trạng thái khác.
- Chi phí hành động (nếu có): đánh giá mức độ tốn kém hoặc hiệu quả của mỗi bước chuyển.

Việc giải quyết bài toán trong AI lúc này tương đương với việc tìm một dãy hành động hợp lệ dẫn từ trạng thái ban đầu đến một trong các trạng thái mục tiêu, sao cho thỏa mãn các ràng buộc về chi phí, thời gian, hoặc tối ưu khác.

Tùy vào cách sử dụng thông tin trong quá trình tìm kiếm, các thuật toán được phân loại thành hai nhóm lớn:

- Tìm kiếm mù (Uninformed Search): không có thông tin bổ sung về trạng thái mục tiêu ngoài việc nhận diện nó.
- Tìm kiếm có thông tin (Informed Search): tận dụng thông tin heuristic để hướng dẫn quá trình tìm kiếm hiệu quả hơn.

Mỗi nhóm có những thuật toán đặc trưng phù hợp với từng loại bài toán cụ thể.

### *1.3.1 Các thuật toán tìm kiếm mù*

#### *1.3.1.1 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 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

***b) 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

}

### *1.3.1.2 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 TG hoặc đã hết các đỉnh để duyệt.

#### *b. 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 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 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

}

### *1.3.2 Các thuật toán tìm kiếm có thông tin*

#### *1.3.2.1. Thuật toán $A^T$*

Là một phương pháp tìm kiếm theo kiểu BeFS 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).

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)

Khi đó đường đi p = n1, n2, …$n_k$ có giá được tính theo công thức:

$$C(p) = \sum_{i=1}^{k-1} C(n_i, n_{i+1})$$

Vấn đề đặt ra là tìm đường đi p từ $T_0$ đến đỉnh $T_G \in$ Goal sao cho c(p)→ min

*Vào*: - Đồ thị G =(V, E)

C: E → $R_+$

e → C(e)

- Đỉnh đầu $T_0$ và Goal chứa tập các đỉnh đích

*Ra*: Đường đi p: $T_0 \rightarrow T_G \in$ Goal sao cho:

$C(p) = g(n_k) = \min \{g(n)/n \in \text{Goal}\}$.

*Phương pháp*: Sử dụng hai danh sách CLOSE và OPEN

```
void AT(){
  OPEN ={T0}, g(T0) = 0, CLOSE = ∅
  while OPEN ≠ ∅ do
  { n ← getNew(OPEN) // Lấy đỉnh n sao cho
  g(n)đạt min
  if (n == TG) return True
  else{
      for (m ∈
        B(n)) do
        if(m ∉ OPEN) && (m ∉ CLOSE)
            then{ g(m)=g(n) + cost(m,n)
        OPEN=OPEN ∪ {m}
```

}else g(m)=min{g(m),$g_{new}$(m)}

CLOSE = CLOSE ∪ {n}

}

return False;

}

### *1.3.2.2. Thuật toán $A^*$*

**Khái niệm**

$A^*$ là một phiên bản đặc biệt của $A^{KT}$ áp dụng cho trường hợp đồ thị. Thuật toán này tìm một đường đi từ một nút 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.

Thuật toán A*

A* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (*priority queue*). Thứ tự ưu tiên gán cho một đường đi  được quyết định bởi hàm .

Trong đó, là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ . Ví dụ, nếu "chi phí" được tính là khoảng cách đã đi qua, khoảng cách đường chim bay giữa hai điểm trên một bản đồ là một đánh giá heuristic cho khoảng cách còn phải đi tiếp.

```
void Asao(){
OPEN = {T0},CLOSE = ⌀,
g(T0) = 0, Tính h(T0), f(T0) = g(T0) + h(T0);
while(OPEN != ⌀)
{
    n = getnew(OPEN)  // lấy đỉnh n sao cho f(n) đạt min.
    if(n == TG) return True;
    else
    {
```

```
        for(m ∈ B(n))
        if(m ∉ (OPEN ∪ CLOSE)){
                Tính h(m), g(m),
                f(m) = g(m) + h(m);
                Cha(m) = n;
                OPEN = OPEN ∪ {m};
            }else
            {
            g(m) = min {g_old(m), g_new(m)};
                Cập nhật lại OPEN;
            }
        }
        CLOSE = CLOSE ∪ {n};
}
return false;
}
```

**Các tính chất**

Cũng như tìm kiếm theo chiều rộng (*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*:

$$h(x) \leq g(y) - g(x) + h(y)$$

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.

**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 A* chính là thuật giải tiêu biểu cho Heuristic.

**Nhược điểm**

A* 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

### *1.3.2.1. Thuật toán $A^{KT}$*

Thuật giải $A^T$ 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) đạt 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 xung 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 đỉnh 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.

# CHƯƠNG II. THUẬT TOÁN BREADTH-FIRST SEARCH VÀ ỨNG DỤNG VÀO BÀI TOÁN ĐONG NƯỚC

## 2.1. Thuật toán Breadth-First Search

### *2.1.1. Nguyên lý hoạt động*

Thuật toán Tìm kiếm theo chiều rộng (BFS) là một phương pháp duyệt đồ thị hoặc cây theo một cách thức có hệ thống, đảm bảo rằng tất cả các nút ở một "cấp độ" nhất định sẽ được thăm trước khi chuyển sang các nút ở cấp độ sâu hơn. Nguyên lý cốt lõi của BFS dựa trên việc sử dụng cấu trúc dữ liệu hàng đợi (Queue), hoạt động theo nguyên tắc FIFO (First-In, First-Out) – phần tử nào vào trước sẽ ra trước.

Quá trình hoạt động của BFS diễn ra như sau:

1. **Khởi tạo**:
    - Bắt đầu từ một nút nguồn (start node) đã chọn. Nút này được đánh dấu là đã thăm và thêm vào hàng đợi.
    - Một tập hợp các nút đã thăm (visited set) được duy trì để theo dõi các nút đã được xử lý, tránh lặp lại và tạo vòng lặp vô hạn.
2. **Duyệt theo chiều rộng**:

    Trong khi hàng đợi không rỗng, thực hiện các bước sau:
    - Dequeue: Lấy (loại bỏ) nút đầu tiên ra khỏi hàng đợi. Gọi nút này là U.
    - Xử lý nút hiện tại: U là nút đang được thăm dò. Ở đây, ta kiểm tra xem U có phải là nút đích (mục tiêu) hay không. Nếu có, quá trình tìm kiếm kết thúc.
    - Thăm các nút kề: Duyệt qua tất cả các nút kề (neighbors) của U.

- Enqueue các nút chưa thăm: Đối với mỗi nút kề V của U mà chưa được thăm:
    - Đánh dấu V là đã thăm.
    - Thêm V vào cuối hàng đợi.

3. **Kết thúc**:
    - Quá trình lặp lại cho đến khi hàng đợi rỗng (tức là tất cả các nút có thể đến được từ nút nguồn đã được thăm) hoặc khi nút đích được tìm thấy.

Nguyên lý FIFO của hàng đợi đảm bảo rằng BFS sẽ khám phá các nút theo thứ tự tăng dần của khoảng cách (số cạnh) từ nút nguồn. Điều này là lý do tại sao BFS luôn tìm thấy đường đi ngắn nhất (ít cạnh nhất) trong các đồ thị không có trọng số.

### 2.1.2. So sánh với các thuật toán khác

Việc so sánh các thuật toán tìm kiếm giúp chúng ta hiểu rõ hơn về điểm mạnh, điểm yếu của từng phương pháp và lý do tại sao BFS lại là lựa chọn phù hợp cho bài toán đong nước. Dưới đây là so sánh chi tiết giữa BFS (Tìm kiếm theo chiều rộng), DFS (Tìm kiếm theo chiều sâu), A* Search và Best-First Search

| Tiêu chí | BFS(Tìm kiếm theo chiều rộng) | DFS(Tìm kiếm theo chiều sâu) | A* Search | Best-First Search(Tìm kiếm tốt nhất tham lam) |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Nguyên lý hoạt động | Duyệt theo chiều rộng. Thăm các nút theo từng cấp độ (từ gần điểm bắt đầu nhất) trước khi đi sâu hơn. Sử dụng hàng đợi (Queue) . | Duyệt theo chiều sâu. Khám phá một nhánh càng sâu càng tốt, sau đó quay lui và khám phá nhánh khác. Sử dụng ngăn xếp (Stack) (hoặc đệ quy). | Duyệt có thông tin. Mở rộng nút dựa trên hàm đánh giá f(n)=g(n)+h(n) (chi phí thực tế từ đầu + ước lượng chi phí tới đích). Sử dụng hàng đợi ưu tiên. | Duyệt có thông tin. Mở rộng nút dựa trên hàm heuristic h(n) (chỉ ước lượng chi phí còn lại tới đích). Sử dụng hàng đợi ưu tiên. |
| Cấu trúc dữ liệu chính | Hàng đợi (Queue) và Tập hợp (Set) các nút đã thăm. | Ngăn xếp (Stack) và Tập hợp (Set) các nút đã thăm. | Hàng đợi ưu tiên (Priority Queue) và Tập hợp (Set) các nút đã thăm. | Hàng đợi ưu tiên (Priority Queue) và Tập hợp ( Set) các nút đã thăm. |
| Tính đầy đủ | Có. Luôn tìm thấy giải pháp nếu tồn tại (trong không gian trạng thái hữu hạn). | Có (nếu có cơ chế kiểm tra nút đã thăm và không có chu trình vô hạn). Nếu không, có thể bị kẹt vô hạn. | Có (nếu hàm heuristic là chấp nhận được). | Không. Có thể không tìm thấy giải pháp ngay cả khi có, do bị kẹt trong các nhánh không tối ưu hoặc chu trình. |

| | | | | |
|---|---|---|---|---|
| Tính tối ưu | Có (trên đồ thị không trọng số). Luôn tìm đường đi ngắn nhất (ít bước nhất). | Không. Có thể tìm thấy đường đi dài hơn. | Có (nếu hàm heuristic là chấp nhận được và nhất quán). | Không. Chỉ tìm con đường "có vẻ tốt nhất" ở thời điểm hiện tại, không đảm bảo là đường đi ngắn nhất. |
| Độ phức tạp thời gian | Tỷ lệ với số nút tối đa trên một cấp độ nhân với số nhánh trung bình (khoảng b mũ d, với d là độ sâu của lời giải). | Tỷ lệ với số nhánh trung bình mũ độ sâu tối đa của đồ thị (b mũ m, với m là độ sâu tối đa của không gian trạng thái). | Phụ thuộc vào chất lượng của hàm heuristic. Với heuristic tốt, có thể nhanh hơn BFS/DFS rất nhiều. | Phụ thuộc vào chất lượng của hàm heuristic. Có thể rất nhanh, nhưng cũng có thể rất chậm nếu heuristic dẫn sai đường. |
| Độ phức tạp không gian | Tỷ lệ với số nút tối đa trên một cấp độ (b mũ d). Có thể tốn nhiều bộ nhớ cho đồ thị rộng, nông. | Tỷ lệ với số nhánh trung bình nhân với độ sâu tối đa của đồ thị (b nhân m). Thường ít tốn bộ nhớ hơn BFS cho đồ thị sâu. | Phụ thuộc vào chất lượng của hàm heuristic. Có thể tốn nhiều bộ nhớ nếu nhiều nút được giữ trong hàng đợi ưu tiên. | Phụ thuộc vào chất lượng của hàm heuristic. Tương tự A*, có thể tốn bộ nhớ nếu hàng đợi ưu tiên lớn. |

| | | | | |
|---|---|---|---|---|
| Ưu điểm | - Đảm bảo tìm đường đi ngắn nhất (ít bước nhất<br>- Dễ cài đặt và hiểu. | - Ít tốn bộ nhớ hơn BFS cho các đồ thị rất sâu.<br>– Thích hợp cho việc kiểm tra sự tồn tại của một đường đi. | - Rất hiệu quả và tối ưu với heuristic tốt.<br>- Định hướng tìm kiếm hiệu quả hơn các thuật toán không thông tin. | - Có thể tìm thấy giải pháp rất nhanh nếu heuristic dẫn đường cực kỳ tốt, vì nó ít phải tính toán chi phí thực tế. |
| Nhược điểm | - Tốn nhiều bộ nhớ (đặc biệt cho đồ thị rộng và nông).<br>- Có thể chậm khi lời giải nằm ở độ sâu rất lớn. | - Không tối ưu (có thể tìm thấy đường đi dài hơn).<br>- Có thể bị kẹt trong nhánh sâu vô hạn hoặc chu trình nếu không có cơ chế kiểm tra. | - Phức tạp hơn để cài đặt<br>- Yêu cầu định nghĩa một hàm heuristic tốt và chấp nhận được. | - Không tối ưu và không đầy đủ.<br>- Dễ bị lạc lối nếu heuristic không chính xác.<br>- Cần định nghĩa một hàm heuristic. |
| Phù hợp với bài toán đong nước | Rất phù hợp. Đảm bảo tìm ra giải pháp với số bước ít nhất. | Ít phù hợp hơn. Có thể tìm được giải pháp nhưng không đảm bảo tối ưu. | Phù hợp nếu có heuristic tốt. Có thể nhanh hơn BFS, nhưng cần nỗ lực để xây dựng heuristic. | Ít phù hợp. Không đảm bảo tối ưu, có thể bỏ lỡ giải pháp ngắn nhất hoặc không tìm thấy giải pháp. |

Từ bảng so sánh chi tiết giữa BFS, DFS, A* Search và Best-First Search, có thể rút ra những điểm kết luận quan trọng về việc lựa chọn thuật toán cho bài toán đong nước:

- BFS nổi bật về tính tối ưu và đầy đủ trong bài toán không trọng số: So với DFS, BFS có ưu điểm vượt trội là luôn tìm thấy đường đi ngắn nhất (tức là số bước đong nước ít nhất) nếu có lời giải. Trong khi đó, DFS không đảm bảo tính tối ưu và có thể tìm ra một lời giải dài hơn. Tính đầy đủ của BFS cũng đáng tin cậy hơn DFS trong trường hợp có chu trình.

- Cân bằng giữa hiệu quả và độ phức tạp: Mặc dù A* có tiềm năng hiệu quả hơn về thời gian với một heuristic tốt, nhưng BFS lại đơn giản hơn trong cài đặt và không yêu cầu xây dựng hàm heuristic. Đối với bài toán đong nước, nơi chi phí của mỗi bước là như nhau và không gian trạng thái không quá lớn đến mức BFS trở nên quá tốn bộ nhớ, việc đảm bảo tính tối ưu mà không cần heuristic phức tạp là một lợi thế lớn của BFS.

- Đáng tin cậy hơn Best-First Search: Best-First Search (Greedy Best-First Search) tuy có thể nhanh chóng tiếp cận mục tiêu nếu heuristic rất tốt, nhưng lại không đảm bảo tính đầy đủ hay tối ưu. Điều này có nghĩa là nó có thể bỏ lỡ lời giải ngắn nhất hoặc thậm chí không tìm thấy lời giải nào. BFS, với tính đảm bảo và tối ưu của mình, mang lại sự chắc chắn hơn nhiều.

Tóm lại, BFS là lựa chọn lý tưởng cho bài toán đong nước nhờ khả năng đảm bảo tìm được lời giải tối ưu (ít bước nhất) và tính đầy đủ, đồng thời sở hữu cấu trúc cài đặt tương đối đơn giản so với các thuật toán tìm kiếm có thông tin phức tạp hơn.

## 2.2. Áp dụng vào bài toán Đong nước

### *2.2.1. Mô tả bài toán và yêu cầu đầu ra*

**Mô tả bài toán:**

Bài toán đong nước là một bài toán kinh điển trong lĩnh vực trí tuệ nhân tạo và được biểu diễn dưới dạng tìm kiếm trong không gian trạng thái. Cụ thể, bài toán có thể được mô tả như sau:

Cho hai bình nước có dung tích lần lượt là m lít và n lít, và một nguồn nước không giới hạn. Mục tiêu là sử dụng các thao tác đổ nước giữa hai bình, đổ nước đi hoặc lấy nước từ nguồn để đạt được đúng k lít nước trong một trong hai bình.

**Ví dụ kinh điển**:
Cho hai bình có dung tích 4 lít và 3 lít, làm sao để đong được chính xác 2 lít nước?

**Các thao tác hợp lệ bao gồm:**

- Đổ đầy một bình từ nguồn nước.
- Rót nước từ bình này sang bình kia. Đổ bỏ nước trong một bình.
- Bình 1 với dung tích tối đa là **m**
- Bình 2 với dung tích tối đa là **n**

Mỗi trạng thái của hệ thống được biểu diễn dưới dạng **(x, y)**, trong đó:

- **x** là lượng nước hiện có trong bình 1
- **y** là lượng nước hiện có trong bình 2

**Các phép toán hợp lệ gồm:**

1. Đổ đầy bình 1

$(x, y) \rightarrow (m, y)$

2. Đổ đầy bình 2

$(x, y) \rightarrow (x, n)$

3. Đổ hết nước trong bình 1

$(x, y) \rightarrow (0, y)$

4. Đổ hết nước trong bình 2

$(x, y) \rightarrow (x, 0)$

5. Rót nước từ bình 1 sang bình 2

- Nếu $x + y \leq n$:
  $(x, y) \rightarrow (0, x + y)$
- Nếu $x + y > n$:
  $(x, y) \rightarrow (x - (n - y), n)$

6. Rót nước từ bình 2 sang bình 1

Nếu $x + y \leq m$:
$(x, y) \rightarrow (x + y, 0)$

Nếu $x + y > m$:
$(x, y) \rightarrow (m, y - (m - x))$

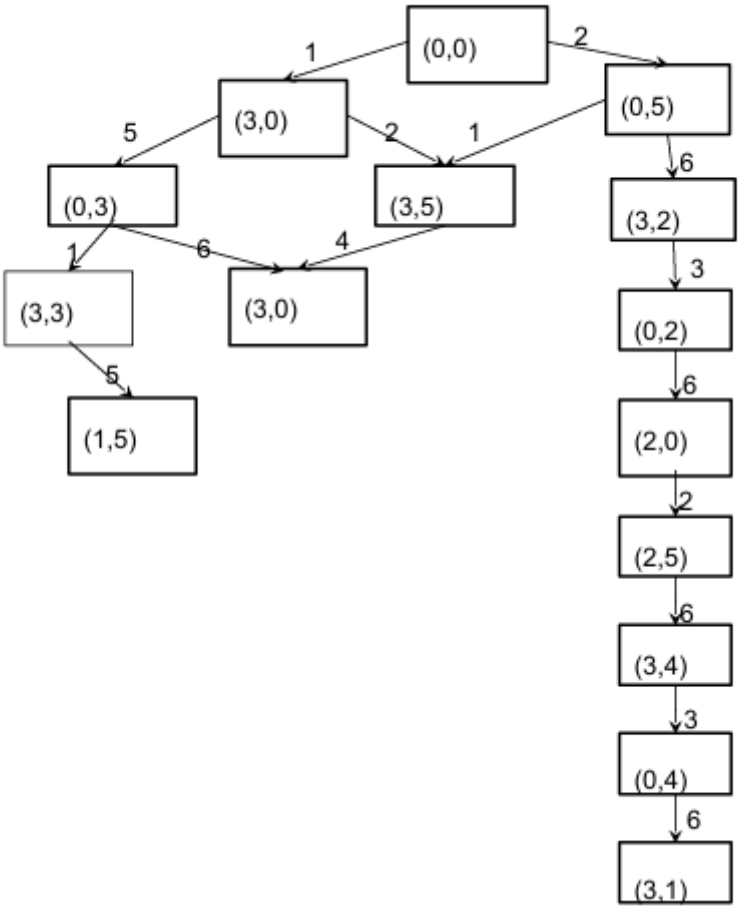
### *2.2.2. Không gian trạng thái*

Bài toán rót nước: Cho 2 bình nước có dung tích là 3 và 5 lít. Làm thế nào để đong được 1 lít.

Trạng thái ban đầu: (0,0)

Trạng thái cuối: (-, 1)

**Mô tả không gian trạng thái của bài toán:**




*Hình 2.1: Không gian trạng thái*

# CHƯƠNG III. CÀI ĐẶT VÀ THỰC NGHIỆM

## 3.1. Ngôn ngữ lập trình và môi trường thực hiện

Trong quá trình xây dựng chương trình giải bài toán đong nước, nhóm đã lựa chọn ngôn ngữ lập trình Python để triển khai các thuật toán. Python là một ngôn ngữ bậc cao, linh hoạt, cú pháp đơn giản và có thư viện hỗ trợ phong phú, rất phù hợp cho việc thử nghiệm, mô phỏng và trực quan hóa các thuật toán tìm kiếm.

### a. Ngôn ngữ lập trình

- **Python 3.11** được sử dụng làm ngôn ngữ chính cho toàn bộ hệ thống.
- Một số thư viện được dùng hỗ trợ cài đặt và trực quan hóa:
    - collections – hỗ trợ cấu trúc hàng đợi (deque) cho thuật toán BFS.
    - matplotlib – dùng để vẽ đồ thị, biểu diễn kết quả trực quan (nếu có).
    - time – dùng để đo thời gian thực thi của mỗi thuật toán.

b. Môi trường thực hiện

- **Hệ điều hành**: Windows 10 hoặc Ubuntu 20.04
- **Trình thông dịch**: Python 3.10 trở lên
- **Trình soạn thảo mã nguồn**: VS Code hoặc PyCharm
- **Thư viện sử dụng**:
    - collections.deque để cài đặt hàng đợi (queue) cho BFS
    - matplotlib (nếu có yêu cầu vẽ đồ thị minh họa)
    - time để đo thời gian thực thi (nếu cần đánh giá hiệu suất)

Lý do lựa chọn

Việc lựa chọn Python và các công cụ trên giúp đẩy nhanh tiến độ thực hiện, đảm bảo độ chính xác của thuật toán, đồng thời tạo thuận lợi trong việc kiểm tra, minh họa trạng thái và kết quả đầu ra.

### 3.2. Ý tưởng và thiết kế thuật toán

**Cấu trúc dữ liệu sử dụng:**

Để biểu diễn trạng thái và quá trình tìm kiếm lời giải cho bài toán, các cấu trúc dữ liệu sau được sử dụng:

- **Hàng đợi (collections.deque)**:
    - **Mục đích:** Đây là cấu trúc dữ liệu cốt lõi và quan trọng nhất của BFS. Hàng đợi đảm bảo nguyên tắc FIFO, cho phép thuật toán duyệt các trạng thái theo từng "cấp độ" (các trạng thái có cùng số bước từ trạng thái ban đầu) trước khi chuyển sang cấp độ tiếp theo.
    - **Cài đặt trong Python**: collections.deque được sử dụng thay vì list thông thường để làm hàng đợi vì nó cung cấp hiệu suất O(1) cho các thao tác thêm vào cuối (append()) và lấy ra từ đầu (popleft()), điều này rất quan trọng để duy trì hiệu quả cho các đồ thị lớn.
    - **Lưu trữ**: Mỗi phần tử trong hàng đợi không chỉ là trạng thái mà là một **tuple** (current_state, path).
        - current_state: Là một **tuple** (jug1, jug2) biểu diễn lượng nước trong bình 1 và bình 2 tại một thời điểm nhất định. Việc sử dụng tuple (kiểu dữ liệu bất biến) giúp các trạng thái này có thể được sử dụng làm khóa trong tập hợp visited.
        - path: Là một **danh sách (list)** các trạng thái đã đi qua từ trạng thái ban đầu đến current_state. Danh sách này cho phép tái tạo lại chuỗi các bước giải pháp khi tìm thấy mục tiêu.
- **Tập hợp (set)**:

- **Mục đích**: Tập hợp visited được dùng để lưu trữ các trạng thái đã được thăm dò. Vai trò chính của nó là ngăn chặn thuật toán lặp lại các trạng thái đã xử lý trước đó, tránh vòng lặp vô hạn và cải thiện đáng kể hiệu suất bằng cách loại bỏ các phép tính trùng lặp.
- **Cài đặt trong Python**: set của Python cho phép kiểm tra sự tồn tại của một phần tử (current_state not in visited) và thêm phần tử mới (visited.add(current_state)) với độ phức tạp trung bình là O(1). Nhờ đó, việc quản lý các trạng thái đã thăm trở nên rất nhanh chóng.

- **Danh sách (list)**:
  - **Mục đích**: Ngoài việc được sử dụng trong path để lưu trữ chuỗi các trạng thái, danh sách cũng được dùng để thu thập các trạng thái lân cận (neighbors) trong hàm get_neighbors().

**Mô hình dữ liệu tổng quát:**

- Trạng thái: State = (x, y) hoặc State = (x, y, parent)
- Danh sách trạng thái cần duyệt: open_list
- Danh sách trạng thái đã duyệt: closed_set

### 3.3. Mã nguồn chương trình và giải thích

***a. Nhập thư viện***

```
from collections import deque
```

**Deque:** dùng để tạo hàng đợi hiệu quả phục vụ BFS,giúp cài đặt thuật toán BFS hiệu quả hơn bằng cách cho phép thêm và lấy phần tử ở hai đầu với tốc độ nhanh (O(1)).

***b. Nhập dữ liệu đầu vào***

```
# Nhập dung tích bình và số lít muốn đong
capacity_1 = int(input("Nhập dung tích bình 1 (lít): "))
capacity_2 = int(input("Nhập dung tích bình 2 (lít): "))
target_amount = int(input("Nhập số lít nước muốn đong: "))
```

Dùng để nhập dữ liệu đầu vào từ người dùng, cụ thể:

- capacity_1: dung tích của bình 1 (đơn vị: lít)
- capacity_2: dung tích của bình 2 (đơn vị: lít)
- target_amount: số lít nước bạn muốn đong chính xác

***c. Kiểm tra trạng thái đích***

```
# Hàm kiểm tra trạng thái mục tiêu
def is_goal_state(state):
    return target_amount in state  # Chỉ cần một trong hai bình chứa đúng số lít cần đong
```

Hàm này dùng để kiểm tra xem trạng thái hiện tại có đạt mục tiêu không, cụ thể:

- state là một tuple (x, y) – số lít nước trong bình 1 và bình 2.
- return target_amount in state: trả về True nếu một trong hai bình chứa đúng số lít nước cần đong (target_amount), ngược lại trả về False.

***d. Sinh các trạng thái kề (neighbor)***

```python
# Sinh các trạng thái con
def get_neighbors(state, cap1, cap2):
    neighbors = []
    jug1, jug2 = state

    # 1. Đổ đầy bình 1
    if jug1 < cap1:
        neighbors.append((cap1, jug2))
    # 2. Đổ đầy bình 2
    if jug2 < cap2:
        neighbors.append((jug1, cap2))
    # 3. Đổ từ bình 1 sang bình 2
    if jug1 > 0 and jug2 < cap2:
        transfer = min(jug1, cap2 - jug2)
        neighbors.append((jug1 - transfer, jug2 + transfer))
    # 4. Đổ từ bình 2 sang bình 1
    if jug2 > 0 and jug1 < cap1:
        transfer = min(jug2, cap1 - jug1)
        neighbors.append((jug1 + transfer, jug2 - transfer))
    # 5. Đổ bỏ bình 1
    if jug1 > 0:
        neighbors.append((0, jug2))
    # 6. Đổ bỏ bình 2
    if jug2 > 0:
        neighbors.append((jug1, 0))

    return neighbors
```

Hàm này dùng để sinh ra tất cả các trạng thái tiếp theo có thể thực hiện được từ một trạng thái hiện tại, dựa trên 6 thao tác đong nước hợp lệ.

Tham số:

- State: trạng thái hiện tại dưới dạng.

- (jug1, jug2):lượng nước trong mỗi bình

- cap1, cap22: dung tích tối đa của bình 1 và bình 2

Các thao tác được xét:

1. Đổ đầy bình 1
2. Đổ đầy bình 2
3. Rót từ bình 1 sang bình 2 (cho đến khi bình 1 hết hoặc bình 2 đầy)
4. Rót từ bình 2 sang bình 1
5. Đổ hết nước trong bình 1

6. Đổ hết nước trong bình 2

Tác dụng: hàm này trả về danh sách tất cả các trạng thái hợp lệ kế tiếp, dùng cho thuật toán BFS để mở rộng các bước tiếp theo.

***e. Thuật toán BFS***

```
# Giải bằng BFS
def bfs(cap1, cap2):
    initial_state = (0, 0)
    queue = deque([(initial_state, [])])
    visited = set()

    while queue:
        current_state, path = queue.popleft()

        if is_goal_state(current_state):
            return path + [current_state]

        if current_state not in visited:
            visited.add(current_state)

            for neighbor in get_neighbors(current_state, cap1, cap2):
                if neighbor not in visited:
                    queue.append((neighbor, path + [current_state]))

    return None  # Không tìm thấy giải pháp
```

Hàm này cài đặt thuật toán BFS (Breadth-First Search) để tìm chuỗi bước ngắn nhất giúp đong chính xác số lít nước mong muốn bằng hai bình.

**Giải thích chi tiết:**

- initial_state = (0, 0): bắt đầu từ khi cả hai bình đều rỗng
- queue: hàng đợi chứa các cặp (trạng thái hiện tại, đường đi đã đi qua)
- visited: tập lưu các trạng thái đã xét để tránh lặp lại

**Vòng lặp chính:**

- Lặp đến khi hàng đợi trống

- Với mỗi trạng thái trong hàng đợi:
    - Nếu đạt mục tiêu (is_goal_state) → trả về đường đi
    - Nếu chưa xét → đánh dấu đã xét
    - Sinh các trạng thái hàng xóm (get_neighbors) và thêm vào hàng đợi nếu chưa xét

**Kết thúc:**

- Nếu không tìm được lời giải → trả về None

***f. In kết quả***

```
# In kết quả
def print_solution(solution):
    if solution:
        print("\nCác bước thực hiện:")
        for step in solution:
            print(f"Bình 1: {step[0]} lít, Bình 2: {step[1]} lít")
    else:
        print("Không tìm thấy giải pháp!")
```

Hàm này dùng để in ra kết quả lời giải nếu thuật toán BFS tìm được.

**Giải thích:**

- Nếu solution **khác None**:
    - In từng trạng thái trong đường đi
    - Mỗi bước hiển thị số lít nước trong **bình 1** và **bình 2**
- Nếu không tìm được (solution == None):
    - In: "Không tìm thấy giải pháp!"

*g. Đo thời gian thực thi*

```
import time

start_time = time.time()
# Chạy chương trình
solution = bfs(capacity_1, capacity_2)
end_time = time.time()

print_solution(solution)
print(f"\nThời gian thực thi: {end_time - start_time:.4f} giây")
```

Giúp theo dõi hiệu suất thực tế của thuật toán và xem kết quả đong nước một cách rõ ràng.

### 3.4. Kết quả chạy thử và giao diện minh họa

Hình dưới đây minh họa giao diện của chương trình giải bài toán đong nước sử dụng thuật toán tìm kiếm theo chiều rộng (BFS). Giao diện được xây dựng đơn giản, trực quan với các ô nhập liệu cho người dùng và các nút điều khiển chức năng chính.

**Giao diện bao gồm:**

- **Các ô nhập liệu** cho:
    - Dung tích bình 1.
    - Dung tích bình 2.
    - Số lít cần đong.
- **Nút "Tìm giải pháp"**: Khi bấm vào, chương trình sẽ thực thi thuật toán BFS để tìm lời giải (nếu có).
- **Nút "Biểu đồ hiệu suất"**: Dùng để hiển thị trực quan hiệu suất hoạt động (chức năng nâng cao).

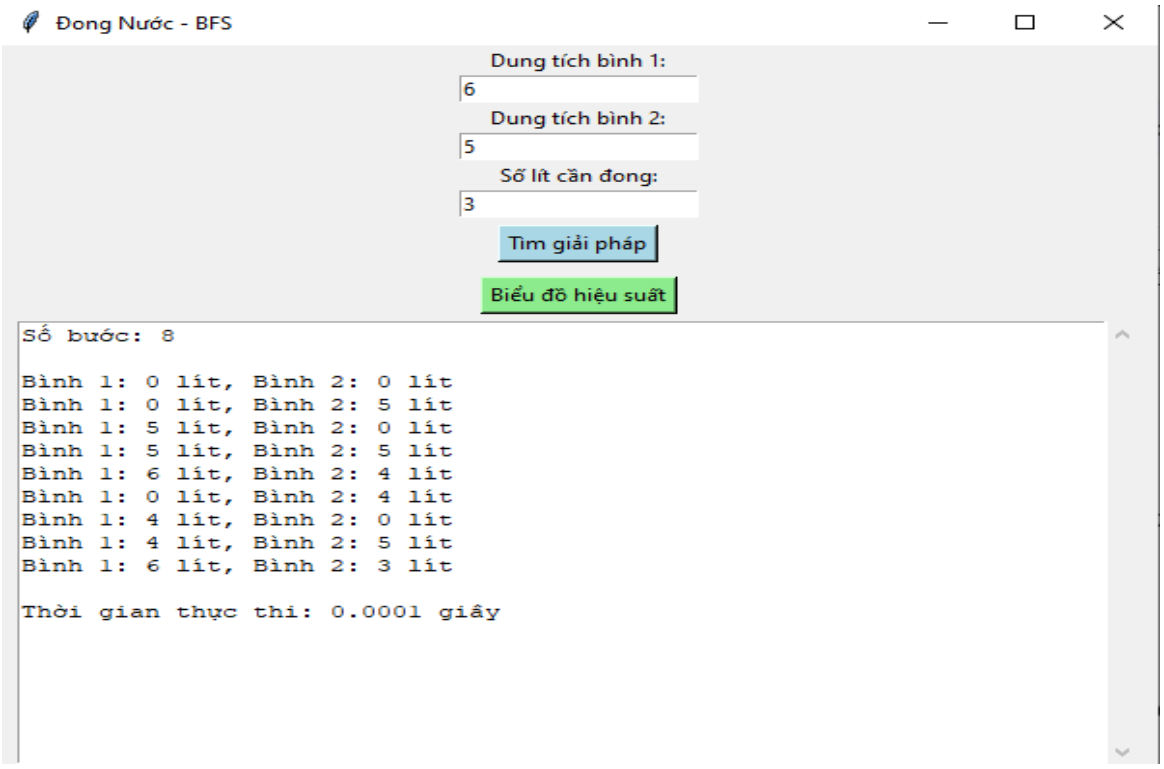
- **Khu vực kết quả**: Hiển thị số bước thực hiện, các trạng thái trung gian của hai bình và thời gian thực thi.

**Ví dụ chạy thử:**

- Dung tích bình 1: 6 lít
- Dung tích bình 2: 5 lít
- Mục tiêu cần đong: 3 lít

**Kết quả hiển thị:**

- Số bước thực hiện: **8 bước**
- Dãy trạng thái của hai bình sau mỗi bước được ghi rõ, giúp người dùng dễ dàng theo dõi quá trình chuyển đổi.
- Thời gian thực thi: **0.0001 giây**, cho thấy thuật toán BFS hoạt động nhanh và hiệu quả trong không gian trạng thái nhỏ.




*Hình 3.1: Giao diện minh họa*

Bộ mẫu thử:

| STT | Dung tích Bình 1 | Dung tích Bình 2 | Số lít cần đong | Kết quả (Trạng thái đích) | Số bước | Thời gian thực thi |
|---|---|---|---|---|---|---|
| 1 | 6 lít | 5 lít | 3 lít | (6, 3) | 8 bước | 0.0001 giây |
| 2 | 4 lít | 3 lít | 2 lít | (4, 2) | 4 bước | 0.0001 giây |
| 3 | 5 lít | 3 lít | 4 lít | (4, 3) | 6 bước | 0.0001 giây |
| 4 | 2 lít | 4 lít | 3 lít | Không tìm thấy | — | 0.0001 giây |

➢ Kết luận: Thuật toán BFS là một phương pháp phù hợp và hiệu quả cho bài toán đong nước, đặc biệt trong trường hợp dung tích bình nhỏ và trung bình. BFS đảm bảo tìm được lời giải ngắn nhất nếu tồn tại, đồng thời hoạt động nhanh và ổn định với không gian trạng thái giới hạn.

Kết quả thử nghiệm cho thấy:

- BFS giải được các bài toán có lời giải chỉ trong vài mili giây.
- Với các trường hợp không có lời giải, thuật toán vẫn dừng chính xác mà không bị lặp vô hạn.
- BFS dễ cài đặt, dễ kiểm soát và trực quan hóa được các bước giải quyết bài toán.

Tuy nhiên, trong trường hợp dung tích bình lớn hoặc không gian trạng thái phức tạp, cần cân nhắc sử dụng các thuật toán tối ưu hơn về bộ nhớ như A* hoặc DFS giới hạn độ sâu.

# KẾT LUẬN

Trong khuôn khổ đề tài " Tìm hiểu  thuật toán Breadth First Search và áp dụng vào bài toán đong nước", nhóm đã tiến hành nghiên cứu, cài đặt và đánh giá khả năng áp dụng của thuật toán BFS trong một bài toán điển hình thuộc lớp bài toán tìm kiếm trong không gian trạng thái.

Thông qua quá trình triển khai, nhóm đã:

- Hiểu rõ được nguyên lý hoạt động của thuật toán Breadth-First Search cũng như ưu điểm của nó trong việc tìm kiếm lời giải ngắn nhất (theo số bước).
- Mô hình hóa bài toán đong nước dưới dạng các trạng thái (x, y) và xây dựng hệ thống các thao tác hợp lệ để chuyển đổi giữa các trạng thái.
- Cài đặt thành công giải thuật trong ngôn ngữ Python, kết hợp với các thư viện tiêu chuẩn như collections và time để đảm bảo hiệu năng và tính trực quan.
- Xây dựng giao diện mô phỏng quá trình đong nước, giúp trực quan hóa từng bước thực hiện của thuật toán.
- Thực hiện đánh giá hiệu năng thuật toán qua nhiều trường hợp mục tiêu khác nhau, ghi nhận số bước giải và thời gian xử lý tương ứng.

Qua quá trình thực hiện đề tài, sinh viên không chỉ củng cố được kiến thức lý thuyết về thuật toán tìm kiếm mà còn nâng cao khả năng tư duy mô hình hóa, kỹ năng lập trình và khả năng áp dụng thuật toán vào các bài toán thực tế. Kết quả của đề tài là minh chứng rõ ràng cho mối liên hệ giữa trí tuệ nhân tạo, tư duy thuật toán, và ứng dụng trong thực tiễn.

Tuy nhiên, do giới hạn về thời gian và phạm vi nghiên cứu, đề tài mới chỉ dừng lại ở bài toán với hai bình, chưa mở rộng ra các biến thể phức tạp hơn (ví dụ ba bình, giới hạn số bước, yếu tố thời gian,...). Đây sẽ là những hướng nghiên cứu tiềm năng cho các đề tài sau.

# TÀI LIỆU THAM KHẢO

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

**[2] Stuart Russell, Peter Norvig**, *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson, 2020.

**[3] Deepak Khemani**, *A First Course in Artificial Intelligence*, McGraw-Hill Education, 2013.

**[4] Elaine Rich, Kevin Knight, Shivashankar B. Nair**, *Artificial Intelligence*, 3rd Edition, Tata McGraw-Hill, 2009.

**[5] Nils J. Nilsson**, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann, 1998.

**[6] ACM Digital Library**, *Graph Search Algorithms and Applications in AI*, https://dl.acm.org, truy cập tháng 6 năm 2025.

**[7] IEEE Xplore**, *Breadth-First Search for State-Space Problems*, https://ieeexplore.ieee.org, truy cập tháng 6 năm 2025.

**[8] Nguyễn Văn Hiệp**, *Ứng dụng thuật toán tìm kiếm trong trí tuệ nhân tạo*, Tạp chí Tin học và Điều khiển học, số 2, 2021.