Ảnh có chứa quảng trường

Mô tả được tạo tự động**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

🙠🙟🕮🙝🙢



**BÁO CÁO ĐỒ ÁN**

**MÔN: TRÍ TUỆ NHÂN TẠO**

**Đề tài:**

**THIẾT KẾ VÀ CÀI ĐẶT CHƯƠNG TRÌNH TRÍ TUỆ NHÂN TẠO VỀ TRÒ CHƠI SOKOBAN**

**GVHD:** *PGS. TS Hoàng Văn Dũng*

**Mã HP:**

**SVTH:** *Nhóm 2*

*Thành phố Hồ Chí Minh, tháng 12 năm 2023*

**DANH SÁCH THÀNH VIÊN THAM GIA ĐỒ ÁN**

***Mã học phần:***

***Đề tài:***  Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban

*----------*

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **HỌ VÀ TÊN THÀNH VIÊN** | **MÃ SỐ SINH VIÊN** | **TỶ LỆ THAM GIA** |
| 1 | Nguyễn Phú Thành | 21110299 | 100% |
| 2 | Nguyễn Văn Hào | 21110175 | 100% |
| 3 | Bùi Quốc Khang | 21110 | 100% |
| 2 | Phạm Hùng Phong | 21110 | 100% |

**Ghi chú:**

Tỷ lệ %: Mức độ phần trăm hoàn thành của từng sinh viên tham gia.

***Trưởng nhóm****:* Bùi Quốc Khang

***Nhận xét của giáo viên***

Tp. Hồ Chí Minh - Tháng 12 năm 2023

# LỜI MỞ ĐẦU

Trong thế giới game hiện đại, sự phát triển của công nghệ trí tuệ nhân tạo (AI) đang mở ra cánh cửa cho những trải nghiệm chơi game mới mẻ và hấp dẫn hơn bao giờ hết. Và một trong những ứng dụng tiêu biểu của công nghệ này là trong trò chơi Sokoban - một trò chơi logic thú vị, đòi hỏi người chơi phải sắp xếp và đẩy các hộp đến vị trí đích. Bằng cách sử dụng các thuật toán và kỹ thuật học máy, trí tuệ nhân tạo có thể tạo ra các chiến lược chơi game thông minh, thách thức người chơi và cung cấp trải nghiệm chơi game độc đáo.

Mặc dù trò chơi Sokoban có vẻ đơn giản, nhưng việc tạo ra một hệ thống trí tuệ nhân tạo hiệu quả để chơi trò chơi này vẫn đòi hỏi sự kết hợp hoàn hảo giữa khả năng lập kế hoạch, quản lý tài nguyên và kỹ năng đưa ra quyết định nhanh chóng. Đối với các nhà phát triển game, sự tích hợp của công nghệ trí tuệ nhân tạo trong Sokoban không chỉ là một thử thách mà còn là cơ hội để cung cấp trải nghiệm chơi game độc đáo và thú vị cho người chơi.

Thông qua việc nghiên cứu chi tiết về ứng dụng của trí tuệ nhân tạo trong trò chơi Sokoban, đề tài nghiên cứu này được thực hiện với hy vọng có thể hiểu rõ hơn về cách mà công nghệ này đóng góp vào sự phát triển của ngành công nghiệp game và cách nó ảnh hưởng đến trải nghiệm chơi game của người chơi.

**LỜI CẢM ƠN**

Lời đầu tiên, nhóm em xin gửi lời cảm ơn chân thành nhất đến ***Thầy Hoàng Văn Dũng*** *–* giảng viên bộ môn ***Trí tuệ nhân tạo*** của chúng em. Trong quá trình học tập và tìm hiểu bộ môn, chúng em đã nhận được sự quan tâm giúp đỡ, hướng dẫn rất tận tình và tâm huyết từ Thầy. Thầy đã giúp chúng em tích lũy thêm nhiều kiến thức để có cái nhìn sâu sắc và hoàn thiện hơn trong lĩnh vực công nghệ thông tin. Để từ đó, ứng dụng những kiến thức mà Thầy truyền tải, nhóm em xin trình bày lại những gì mà mình đã học hỏi được thông qua việc thực hiện đề tài ***“Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban”***.

Kiến thức là vô hạn và sự tiếp nhận kiến thức của bản thân mỗi người luôn tồn tại những hạn chế nhất định. Do đó, trong phạm vi khả năng của bản thân, nhóm em đã rất cố gắng để hoàn thành đề tài một cách tốt nhất. Tuy nhiên, chắc chắn không tránh khỏi những thiếu sót, nhóm chúng em rất mong nhận được sự cảm thông và những ý kiến đóng góp đến từ Thầy để đề tài của nhóm em được hoàn thiện hơn.

Một lần nữa, ***nhóm em xin chân thành cảm ơn Thầy*** đã tận tình hướng dẫn, chỉ bảo các thành viên nhóm em trong suốt quá trình học tập và thực hiện đồ án này.

***Kính chúc Thầy sức khỏe, hạnh phúc thành công trên con đường sự nghiệp giảng dạy.***

**Trân trọng**

***Đại diện nhóm***

Bùi Quốc Khang

# MỤC LỤC

[LỜI MỞ ĐẦU 2](#_bookmark0)

[MỤC LỤC 3](#_bookmark1)

[MỤC LỤC HÌNH ẢNH 5](#_bookmark2)

[MỤC LỤC BẢNG BIỂU 6](#_bookmark3)

[CHƯƠNG 1. MỞ ĐẦU 7](#_bookmark4)

* 1. [Phát biểu về bài toán 7](#_bookmark5)
  2. [Mục đích, yêu cầu thực hiện 8](#_bookmark8)
  3. [Đối tượng và phạm vi nghiên cứu 8](#_bookmark9)
     1. [Đối tượng nghiên cứu 8](#_bookmark10)
     2. [Phạm vi nghiên cứu 9](#_bookmark11)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 1](#_bookmark12)0

[2.1. Công cụ và môi trường lập trình 1](#_bookmark13)0

[CHƯƠNG 3. PHÂN TÍCH, THIẾT KẾ GIẢI PHÁP 11](#_bookmark14)

# MỤC LỤC HÌNH ẢNH

[*Bảng* 1 *. Mô tả ví dụ Hill Climbing* 20](#_bookmark25)

*Bảng* 2 *. Bảng phân công nhiệm vụ*

# CHƯƠNG 1. MỞ ĐẦU

#### Phát biểu về bài toán:

Game Sokoban là một trò chơi logic có tính năng giải đố và là một trong những trò chơi cổ điển và phổ biến trong thể loại này. Đề tài về game Sokoban thường bao gồm các khía cạnh sau:

- Luật chơi cơ bản: Trong Sokoban, người chơi điều khiển một nhân vật (thường là một người công nhân hoặc một nhân vật tương tự) để đẩy các hộp (hoặc các đối tượng khác) vào các ô mục tiêu trên một màn chơi, người chơi chỉ có thể đẩy hộp một cách trái phép, không thể kéo hoặc đẩy nhiều hộp cùng một lúc, mục tiêu của trò chơi là di chuyển tất cả các hộp đến vị trí đích trong số lần bước ít nhất có thể.

- Độ khó: Sokoban được thiết kế với nhiều cấp độ khó khác nhau, từ dễ đến rất khó

- Thể loại: Sokoban thuộc thể loại game puzzle và logic

- Ứng dụng: Sokoban không chỉ là một trò chơi giải trí, mà còn được sử dụng trong lĩnh vực nghiên cứu trí tuệ nhân tạo và tối ưu hóa



***Hình 1****. Minh họa game Sokoban*

Trong quá trình giải bài toàn Đề tài về game Sokoban tập trung vào việc nghiên cứu và phát triển các khía cạnh của trò chơi này, bao gồm cách thiết kế màn chơi, độ khó, ứng dụng trong lĩnh vực giáo dục và nghiên cứu. Sokoban là một ví dụ điển hình về trò chơi logic mang tính thách thức cao và vẫn thu hút sự quan tâm của người chơi trên khắp thế giới.

#### Mục đích, yêu cầu thực hiện:

Đề tài ***Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban*** đặt ra các vấn đề trọng tâm về mục tiêu mà sinh viên nghiên cứu cần thực hiện được bao gồm:

Vấn đề đầu tiên được đưa ra về yếu tố lý thuyết, sinh viên thực hiện cần nghiên cứu chuyên môn, hiểu được tổng quan và khả năng ứng dụng của thuật toán mà nhóm đã lựa chọn BFS, DFS, IDFS, Greedy, UCS, Dijkatra, A\*, Beam Search và Hill Climbing. Song song với công nghệ được lựa chọn để sử dụng, nhóm sinh viên cần tìm hiểu thêm cách xây dựng giao diện đẹp, thân thiện với người dùng.

Vấn đề thứ hai mà đề tài đặt ra là ứng dụng các kiến thức đã tìm hiểu được vào xây dựng một sản phẩm cụ thể, ở đây là **Chương trình tìm kiếm lời giải**, sản phẩm hoàn thành phải là một chương trình, đáp ứng các yêu cầu cơ bản, phục vụ cho người dùng trong việc tìm lời giải cho bài toán và có thể so sánh thời gian giải của các thuật toán khác nhau như thế nào.

#### Đối tượng và phạm vi nghiên cứu:

#### Đối tượng nghiên cứu:

Các thuật toán:

+ Breadth First Search (BFS)

+ Depth First Search (DFS)

+ IDFS

+ Greedy

+ UCS

+ A\*

+ Hill climbing

+ BeamSearch

+ Dijskatra

#### Phạm vi nghiên cứu:

Cách tìm kiếm lời giải cho các thuật toán BFS, DFS, IDFS, Greedy, UCS, Dijkatra, A\*, Beam Search và Hill Climbing cho bài toán Sokoban

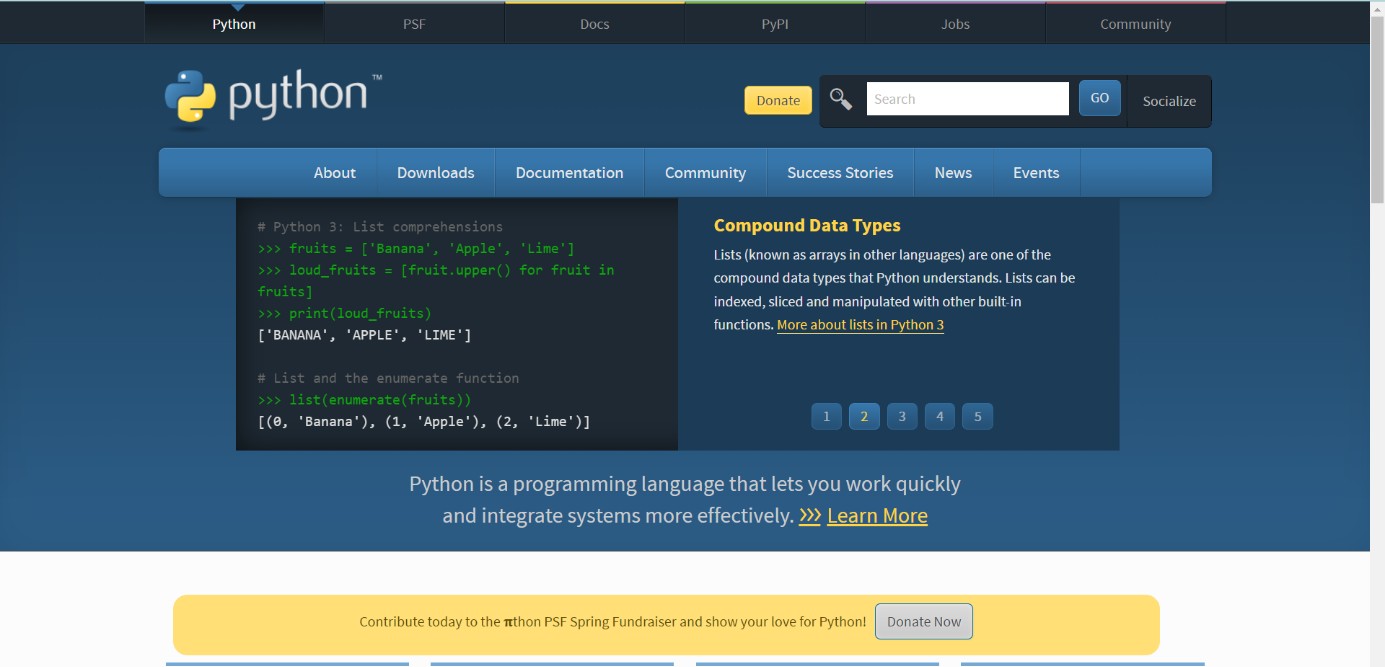
# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

## 2.1 Ngôn ngữ lập trình

### **2.1.1 Ngôn ngữ Python**

Python là một ngôn ngữ lập trình được sử dụng rộng rãi trong các ứng dụng web, phát triển phần mềm, khoa học dữ liệu và máy học (ML). Các nhà phát triển sử dụng Python vì nó hiệu quả, dễ học và có thể chạy trên nhiều nền tảng khác nhau. Phần mềm Python được tải xuống miễn phí, tích hợp tốt với tất cả các loại hệ thống và tăng tốc độ phát triển

Thay vì tích hợp hết tất cả các tính năng vào phần cốt lõi, Python được thiết kế để dễ dàng mở rộng (bằng các mô đun). Tính mô đun nhỏ gọn này đã làm cho Python trở nên phổ biến như là một cách thêm các giao diện lập trình được vào các ứng dụng hiện có. Tầm nhìn của Van Rossum (cha đẻ của Python) về một ngôn ngữ có phần lõi nhỏ với một thứ viện chuẩn rộng lớn và một trình thông dịch dễ dàng mở rộng bắt nguồn từ việc ông nản lòng trước ABC, một ngôn ngữ lập trình tán thành hướng tiếp cận ngược lại.



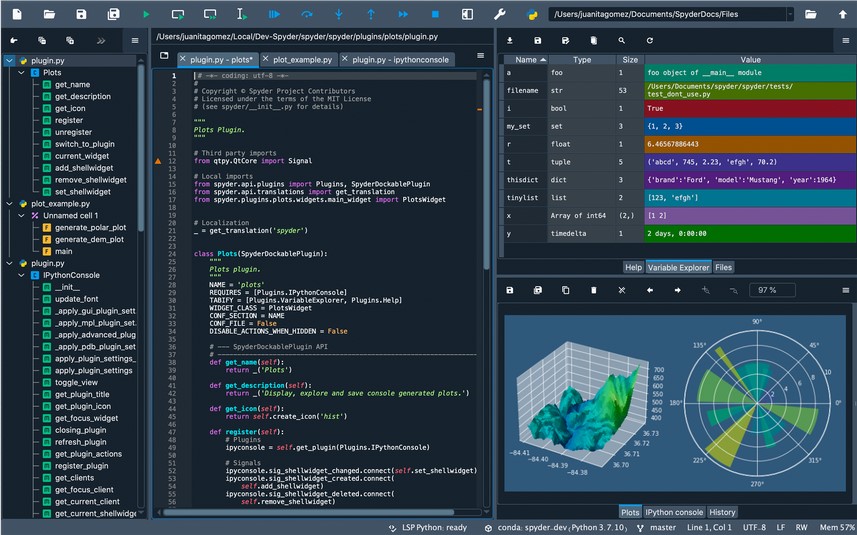
Hình 1. python.org – Trang web chính thức của Python

### **Đặc điểm của ngôn ngữ**

* + - * Python được thông dịch: Python được trình thông dịch xử lý trong thời gian chạy. Bạn không cần phải biên dịch chương trình của mình trước khi thực hiện nó.
      * Python có tính tương tác (Interactive): Tại một dấu nhắc Python (command line) bạn có thể tương tác trực tiếp với trình thông dịch để viết chương trình Python.
      * Python hỗ trợ kỹ thuật lập trình hướng đối tượng hoặc kỹ thuật lập trình đóng gói mã trong các đối tượng.
      * Python là ngôn ngữ linh hoạt và dễ tiếp cận: Python là ngôn ngữ tuyệt vời cho các lập trình viên và hỗ trợ phát triển một loạt các ứng dụng từ xử lý văn bản đơn giản, lập trình web, cho đến lập trình game.
      * Python luôn được xếp hạng vào những ngôn ngữ lập trình phổ biến nhất thế giới.

## 2.2 Môi trường lập trình

Spyder là một môi trường khoa học mã nguồn mở miễn phí được viết bằng Python, dành cho Python, được thiết kế bởi và dành cho các nhà khoa học, kỹ sư và nhà phân tích dữ liệu. Nó có sự kết hợp độc đáo giữa chức năng chỉnh sửa, phân tích, gỡ lỗi và lập hồ sơ nâng cao của một công cụ phát triển toàn diện với khả năng khám phá dữ liệu, thực thi tương tác, kiểm tra sâu và khả năng trực quan hóa của một gói khoa học..

* ***Các ưu điểm của Spyder:***
  + *Môi trường tích hợp*: Spyder kết hợp một loạt chức năng quan trọng như chỉnh sửa mã, phân tích số liệu, gỡ lỗi và trực quan hóa vào một giao diện duy nhất. Điều này giúp người dùng tiết kiệm thời gian và năng lực trong quá trình làm việc.
  + *Dành riêng cho Python*: Spyder tập trung vào ngôn ngữ lập trình Python, tạo điều kiện thuận lợi cho việc phát triển và thử nghiệm mã nguồn Python một cách hiệu quả.
  + *Khám phá dữ liệu tiện lợi*: Spyder cung cấp các công cụ mạnh mẽ cho việc khám phá dữ liệu, giúp người dùng nhanh chóng hiểu rõ thông tin và xu hướng bên trong dữ liệu.
  + *Tương tác và thực thi dễ dàng*: Môi trường tương tác của Spyder cho phép người dùng thực thi từng phần mã và kiểm tra kết quả ngay lập tức, giúp tối ưu quá trình phát triển và chỉnh sửa mã.
  + *Gỡ lỗi thông minh*: Spyder cung cấp các công cụ gỡ lỗi tiện lợi giúp người dùng xác định và sửa lỗi một cách nhanh chóng, giúp tăng hiệu suất làm việc.
  + *Trực quan hóa đẹp mắt*: Với khả năng trực quan hóa dữ liệu mạnh mẽ, Spyder giúp người dùng tạo ra các biểu đồ và đồ thị hấp dẫn để trình bày kết quả nghiên cứu một cách rõ ràng và dễ hiểu.
  + *Phù hợp cho mọi người*: Dù bạn là nhà khoa học, kỹ sư hay nhà phân tích dữ liệu, Spyder đều cung cấp môi trường làm việc linh hoạt và hiệu quả cho tất cả các dự án và nhiệm vụ.
  + 

Hình 2. Giao diện Spyder

## 2.3 Xây dựng giao diện với Pygame

Pygame là một bộ mô-đun Python đa nền tảng được thiết kế để viết trò chơi điện tử. Nó bao gồm đồ họa máy tính và thư viện âm thanh được thiết kế để sử dụng với ngôn ngữ lập trình Python.

Pygame sử dụng thư viện Simple DirectMedia Layer (SDL), với mục đích cho phép phát triển trò chơi máy tính trong thời gian thực mà không cần cơ chế bậc thấp của ngôn ngữ lập trình C và các dẫn xuất của nó. Điều này dựa trên giả định rằng các chức năng đắt tiền nhất bên trong trò chơi có thể được trừu tượng hóa khỏi logic trò chơi, do đó có thể sử dụng ngôn ngữ lập trình bậc cao, chẳng hạn như Python, để cấu trúc trò chơi.

Các tính năng khác mà SDL không có bao gồm toán học vectơ, phát hiện va chạm, quản lý đồ họa 2D, hỗ trợ MIDI, camera, thao tác mảng pixel, chuyển đổi, lọc, hỗ trợ phông chữ freetype nâng cao và vẽ.

Trong đề tài này, nhóm chúng em sử dụng Pygame để xây dựng giao diện và mô

phỏng chuyển động thuật toán.

Đầu mỗi file, khai báo thư viện pygame qua câu lệnh **‘import pygame’**.

# CHƯƠNG 3. PHÂN TÍCH, THIẾT KẾ GIẢI PHÁP

#### Sơ đồ khối, ý tưởng thuật toán:

#### Thuật toán Breadth First Search (BFS):

#### Giới thiệu chung:

Breadth First Search (BFS) là một trong những thuật toán tìm kiếm cơ bản và thiết yếu của đồ thị. Thuật toán tìm đường đi từ đỉnh xuất phát đến đỉnh kết thúc bằng cách duyệt theo chiều rộng.

Thuật toán này được xếp vào nhóm thuật toán tìm kiếm mù hay tìm kiếm không có thông tin. Thuật toán tìm kiếm mù là thuật toán không quan tâm đến trọng số đường đi mà chỉ duyệt những đỉnh kề liên tiếp nó.

Thuật toán BFS thực hiện duyệt các đỉnh từ đỉnh xuất phát và duyệt tiếp tới các đỉnh kề nó, tiếp tục cho đến khi không còn đỉnh nào để đi. Bên cạnh đó trong quá trình duyệt, ta lưu lại các đỉnh đã đi qua để từ đó truy vết tìm ra đường đi ngắn nhất.

* Ưu điểm:
  + Dễ cài đặt.
  + Nếu số đỉnh là hữu hạn và số đỉnh nhỏ thì thuật toán sẽ tìm ra kết quả nhanh chóng.
* Nhược điểm:
  + Vì duyệt qua hầu hết các đỉnh nên nó mang tính chất vét cạn, không nên áp dụng nếu duyệt qua số đỉnh quá lớn.

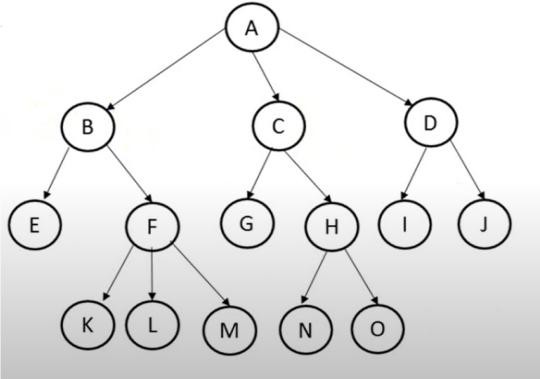
Chính vì là tìm kiếm mù, không chú ý đến thông tin đường đi do đến dẫn đến duyệt qua các đỉnh một cách mù quáng và gây tốn thời gian, bộ nhớ.

 **Ý tưởng thuật toán:**

***Mô tả thuật toán:*** Lưu các đỉnh kề thành 1 danh sách và lấy từng phần tử trong danh sách các đỉnh kề ra để xét. Khi thêm một phần tử đỉnh kề vào danh sách thì phần tử đó sẽ được thêm ở cuối danh sách, còn lấy phần tử ra thì sẽ lấy ra ở đầu danh sách. Danh sách hoạt động như thế này còn được gọi là hàng đợi (queue).



Quá trình biến đổi của danh sách lưu trữ:

Gọi hàng đợi Open là danh sách lưu trữ Thêm A vào hàng đợi

Open=[A]

Lấy phần tử đỉnh A ra xét, duyệt các đỉnh kề

A theo thứ tự từ trái sang phải và thêm vào Open:

Open=[B, C, D]

Lấy phần tử đỉnh B ra xét, duyệt các đỉnh kề B theo thứ tự từ trái sang phải:

Open=[C, D, E, F]

Và cứ tiếp tục như vậy cho đến đỉnh cần tìm.

#### Thuật toán A\*:

Trong khoa học máy tính, A \* là thuật toán tìm kiếm trong đồ thị. Thuật toán tìm một đường dẫn từ nút bắt đầu đến một nút kết thúc đã cho (hoặc đến một nút thỏa mãn điều kiện cho trước). Thuật toán sử dụng hàm "heuristic" để sắp xếp loại của từng nút dựa trên ước tính về tuyến đường tốt nhất qua nút đó. Thuật toán duyệt qua các nút theo thứ tự heuristic này. Do đó, thuật toán A \* là một ví dụ về tìm kiếm tối ưu nhất.

Bài toán tìm đường đi ngắn nhất - bài toán thường dùng A\* để giải. Thuật toán xây dựng tất cả các đường đi tăng dần từ điểm bắt đầu cho tới khi nó tìm thấy một đường đi chạm tới đích đến. Để xác định đường đi nào có khả năng sẽ đi đến đích, A\* sẽ dùng một hàm "heuristic" về khoảng cách từ một điểm bất kỳ cho trước để đến đích. Trong bài toán này, 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.

Sự khác biệt của A\* với tìm kiếm theo lựa chọn tối ưu nhất là nó còn tính đến khoảng cách đã đi được. Điều này làm cho A\* "đầy đủ" và "tối ưu", có nghĩa là, A\* có thể luôn luôn tìm thấy một đường đi ngắn nhất nếu tồn tại đường đi như vậy. Trong một môi trường tìm kiếm mê cung, cách duy nhất để tới đích có thể là trước hết phải đi về phía xa đích đến và cuối cùng là quay lại. Trong đó, việc thử lần lượt các nút theo thứ tự "gần đích thì được thử trước" có thể gây mất nhiều thời gian.

Thuật toán A\* lưu giữ một tập hợp các lời giải chưa được hoàn chỉnh, có nghĩa là những đường đi qua đồ thị cho trước, bắt đầu từ nút xuất phát. Để tính được đầy đủ thông tin đánh giá heuristic cho mỗi trạng thái trong không gian tìm kiếm của bài toán, một hàm đánh giá heuristic f(n) được định nghĩa đó là tổng của hai yếu tố:

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

Trong đó:

+ g(n) là chi phí tính từ trạng thái xuất phát đến trạng thái hiện tại đang xét đến.

+ h(n) là hàm heuristic ước lượng chi phí đi từ trạng thái hiện tại tới đích.

Hàm f(n) có giá trị càng thấp thì trạng thái đó càng có độ ưu tiên và triển vọng cao.



***Hình 3******.*** *Mô tả thuật toán A\**

#### Với h được tính theo hàm Missplace:

Đây là hàm heuristic đơn giản nhất, thông tin đánh giá heuristic của bài toán là số ô đặt không đúng vị trí tại mỗi trạng thái khi nó được so sánh với trạng thái đích.

Trạng thái nào có ít ô đặt không đúng chỗ nhất so với trạng thái đích đó là trạng thái tốt nhất được chọn duyệt qua để tiến đến đích. Nếu có hai trạng thái có cùng thông tin đánh giá heuristic, thì lúc này trạng thái được chọn để duyệt là trạng thái gần trạng thái gốc nhất của đồ thị. Trạng thái này sẽ có đường ngắn nhất để đi đến đích.

Tuy nhiên heuristic này không sử dụng hết các thông tin trong một cấu hình bàn cờ vì nó không đưa vào khoảng cách mà các ô sai khác.

- **Mả giả:**

* + - * 1. Mảng n x n (TH 8-puzzle n = 3)
        2. Duyệt từng phần tử (2 vòng for)

Nếu vị trí hiện tại khác vị trí đích

h = h +1

* + - * 1. Trả lại h.

#### Với h được tính theo hàm Euclid:

Hàm heuristic này sử dụng tính khoảng cách Euclid để tìm ra đường đi thẳng nhanh nhất từ trạng thái đầu đến trạng thái đích. Trong toán học khoảng cách Euclid là độ dài của đoạn thẳng nối hai điểm đó.

Ta có trạng thái đầu sẽ có tọa độ là (i, j) và trạng thái cần đi đến hay chính là trạng thái đích sẽ có tọa độ là (x, y). Có thể suy ra:

Công thức:

(i − x)2 + (j − y)2

Đối với bài toán là những số nằm trong mảng 2 chiều xét theo toán học ta sẽ có công thức trên bằng cách áp dụng định lý Pythagoras cho một tam giác vuông có hai cạnh góc vuông song song với hai trục tọa độ và cạnh huyền là đoạn thẳng nối hai điểm x và y.

#### Ưu điểm:

* + Phổ biến, dễ hiểu, dễ thực hiện, kết quả tốt trong nhiều trường hợp.
  + Đặc biệt Euclid sẽ hiệu quả hơn với dữ liệu ít chiều.

#### Nhược điểm:

* + Khi gặp tọa độ phức tạp ta cần làm đơn giản nó để quá trình tính toán diễn ra nhanh hơn.
  + Khi số chiều tăng lên thì Euclid sẽ kém hiệu quả.
    - 1. **Với h được tính theo hàm Manhattan:**

Khoảng cách Manhattan là thước đo khoảng cách giữa hai điểm trong không gian vectơ N chiều. Nó là tổng độ dài của các hình chiếu của đoạn thẳng giữa các điểm lên các trục tọa độ. Nói một cách dễ hiểu, nó là tổng chênh lệch tuyệt đối giữa các số đo theo mọi chiều của hai điểm.

Nó còn được gọi là định mức L1 nói dễ hiểu hơn hay nó là khoảng cách trong thành phố, là một dạng khoảng cách giữa hai điểm nằm trong không gian với một cặp số tọa độ.

vuông. Ví dụ:

#### Thuật toán Hill Climbing:

#### Khái niệm:

Giải thuật Hill Climbing (giải thuật leo đồi) là một kỹ thuật tối ưu toán học theo phương pháp tìm kiếm cục bộ. Nó thực hiện tìm một trạng thái tốt hơn trạng thái hiện

tại để tiến đến. Để biết trạng thái tiếp theo nào là lớn hơn, nó dùng một hàm heuristic h để xác định trạng thái nào là tốt nhất.

Giải thuật Hill Climbing dễ dàng tìm thấy một giải pháp tốt cục bộ (local optimum) nhưng khó tìm thấy giải pháp tối ưu (global optinum) trong tất cả các giải pháp được đưa ra (search space). Hill Climbing phù hợp để giải các bài toán “convex” (dịch tạm: lồi) như là tìm kiếm đơn giản trong lập trình tuyến tính, tìm kiếm nhị phân.

- **Mã giả:**

* + - * 1. Khởi tạo danh sách L chỉ chứa mảng ban đầu.
        2. Khởi động vòng lặp

Nếu L rỗng => thông báo tìm kiếm thất bại.

Loại trạng thái u ở đầu danh sách.

Nếu u là trạng thái kết thúc => thông báo tìm kiếm thành công và STOP.

Duyệt các trạng thái v kề u và đặt v vào L sao cho các phần tử đưa vào được đánh giá giảm dần (danh sách L luôn giảm).

#### Vấn đề:

*Tối ưu toàn cục (global optimum)*

Rơi vào vùng tối ưu cục bộ là vấn đề lớn nhất mà giải thuật Hill Climbing gặp phải, đó là lúc chúng ta leo lên một đỉnh mà chúng ta không thể tìm thấy đỉnh nào tốt hơn được nữa nhưng đỉnh này lại chưa phải là đỉnh cao nhất.

Để giải quyết vấn đề này, khi leo đến một đỉnh tối ưu cục bộ, để tiếp tục tìm kiếm lời giải tốt hơn nữa ta có thể lặp lại thuật toán Hill Climbing vớí nhiều điểm xuất phát khác nhau được chọn ngẫu nhiên và lưu lại kết quả tốt nhất ở mỗi lần lặp. Nếu số lần lặp đủ lớn thì ta có thể tìm được kết quả tối ưu, tuy nhiên với những bài toán có không gian tìm kiếm khổng lồ (chẳng bạn như bài toán xếp lịch) ta không thể đưa ra số lần lặp đủ lớn để đảm báo tìm được lời giải tối ưu. Cho nên giải thuật leo đồi là phương pháp giải không mang lại nhiều hiệu quả trong trường hợp này.

Như vậy “bề mặt” của không gian tìm kiếm vô cùng quan trọng đối với hiệu quả của bài toán. Nếu bài toán chỉ có lượng đỉnh tối ưu cục bộ nhỏ thì giải thuật sẽ tìm ra lời giải rất nhanh. Tuy nhiên, trong trường hợp không gian tìm kiếm quá lồi lõm, giải thuật sẽ bị loay hoay trong vùng tối ưu cục bộ và không tìm ra lời giải tối ưu của bài toán.

*Chóp nhọn (Ridges and Alleys)*

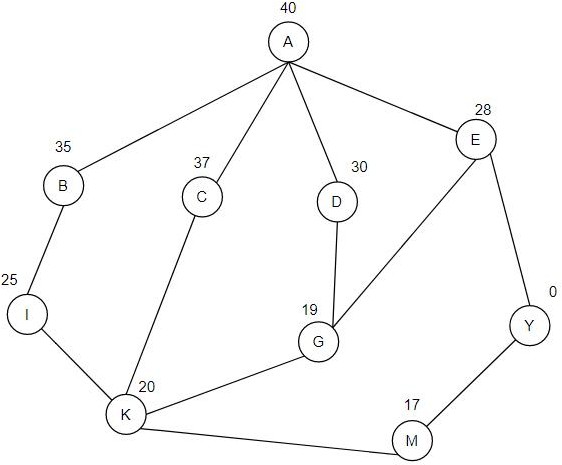
Ridges là một vấn đề thách thức cho các lập trình viên để tối ưu hóa trong không gian liên tục. Khi vị trí hiện tại cao hơn các khu vực lân cận nhưng lại có một độ dốc lớn kiến nó không thể di chuyển trong 1 bước lớn, dẫn đến “nhà leo núi” phải thực hiện các bước rất nhỏ để hướng đến một vị trí tốt hơn. Như vậy có thể mất khoảng thời gian khổng lồ để đến được đích.

*Cao nguyên (Plateau)*

Một vấn đề khác mà đôi khi xảy ra với Hill Climbing là của một cao nguyên. Một cao nguyên hiểu là không gian tìm kiếm là bằng phẳng, các trạng thái lân cận của trạng thái hiện tại đều có cùng một giá trị khiến cho thuật toán không tìm được hướng đi tốt nhất để di chuyển và có thể đi lang thang trong một hướng và thất lạc.

* **Ưu điểm***:* Nhanh, đơn giản, hiệu quả trong không gian tìm kiếm nhỏ, ít lồi lõm.
* **Nhược điểm***:* Khó tìm thấy trạng thái đích nếu như không gian tìm kiếm lớn, có nhiều điểm tối ưu cục bộ.

***Ví dụ về giải thuật Hill Climbing***



***Hình*** ***4.*** *Ví dụ Hill Climbing*

#### input:

h(A)=40,h(B)=35, h(C)=37, h(D)=30, h(E)=28, h(1)=25, h(G)=19, h(K)=17,h(Y)=0.

#### output:

Đường di ngắn nhất từ A đến Y.

#### \* Giải quyết

Bước 1: Cho điểm xuất phát vào trong open ta được open={ A40}.

Bước 2: Lấy đỉnh đầu của open là A40… ra và xét các cạnh kề với A40 ta có các đỉnh B35,C37,D30,E28. Sắp xếp theo thứ tự tăng dần rồi đưa chúng vào L ta được L={ E28,D30,B35,C37}.

Bước 3: Lấy đỉnh đầu của open là E28 ra và xét các cạnh kề với E28 ta có các cạnh G19,Y0 sắp xếp theo thứ tự tăng dần rồi đưa chúng vào hàm L ta được L = { Y0,G19, D30,B35,C37}.

Bước 4:lấy đỉnh đầu của open là Y0 mà Y là đích nên tìm kiếm kết thúc.

***Bảng 1******.*** *Mô tả ví dụ Hill Climbing*

|  |  |  |  |
| --- | --- | --- | --- |
| Lặp | U | Kề U | L |
| 1 |  |  | A40 |
| 2 | A40 | B35,C37,D30,E28 | E28,D30,B35,C37 |
| 3 | E28 | G19,Y0 | Y0,G19, D30,B35,C37 |
| 4 | Y0 |  | G19, D30,B35,C37 |

#### Chi tiết về thuật toán:

#### Thuật toán Breadth First Search (BFS):

* Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
* Close: tập các trạng thái đã được duyệt, ban đầu close rỗng

**Bước 1:** Cho đỉnh xuất phát vào Open.

**Bước 2:** Nếu Open rỗng thì tìm kiếm thất bại, kết thúc tìm kiếm.

**Bước 3:** Lấy phần tử đầu tiên trong Open ra và gán cho biến O. Sau đó cho O vào Close.

**Bước 4:** Kiểm tra nếu O là trạng thái đích thì tìm kiếm thành công và truy vết tìm đường đi. Kết thúc tìm kiếm.

**Bước 5:** Tìm tất cả các trạng thái con của O chưa được duyệt qua thì cho vào cuối Open.

**Bước 6:** Trở lại bước 2.

#### Thuật toán A\*:

* Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
* Close: tập các trạng thái đã được xét đến, ban đầu close rỗng
* n tùy theo hàm heuristic:

+ n = 0: Missplace

+ n = 1: Euclid

+ n = 2: Manhattan

**Bước 1:** Cho đỉnh xuất phát vào Open.

List<State> Open = new List<State>(); List<State> Close = new List<State>();

S.g = 0;

S.h = Hx(S, G, n); Thêm S vào Open

**Bước 2:** Nếu Open rỗng thì tìm kiếm thất bại, kết thúc tìm kiếm.

**Bước 3:** Lấy phần tử đầu tiên trong Open ra và gán cho biến O. Sau đó cho O vào Close.

**Bước 4:** Kiểm tra nếu O là trạng thái đích thì tìm kiếm thành công và truy vết tìm đường đi. Kết thúc tìm kiếm.

**Bước 5:** Tìm tất cả các trạng thái con của O chưa được duyệt qua thì cho vào cuối Open.

for (int i = 0; i < 4; i++)

+Sinh ra các trạng thái con của O

+Gọi child là con của O, nếu child không nằm trong Open và Close thì thêm child vào Open theo thứ tự tăng dần đối với hàm đánh giá.

**Bước 6:**Trở lại bước 2.

#### Với h được tính theo hàm Missplace:

int Hmiss(S, G)

{

cost = số vị trí khác nhau giữa trạng thái bắt đầu và trạng thái đích. return cost

}

Trong đó cost là chi phí.

#### Với h được tính theo hàm Euclid:

int Heuclid(S,G)

{

thái đích.

(i − x)2 + (j − y)2

}

cost = khoảng cách return cost

từ trạng thái bắt đầu đến trạng

Trong đó: + cost là chi phí.

+ (i,j) là tọa độ của trạng thái bắt đầu.

+ (x,y) là tọa độ của trạng thái kết thúc.

#### Với h được tính theo hàm Manhattan:

int Hmanhattan(S,G)

{

cost = khoảng cách |(i-x) + (j-y)| từ trạng thái bắt đầu đến trạng thái đích. return cost

}

Trong đó: + cost là chi phí.

+ (i,j) là tọa độ của trạng thái bắt đầu.

+ (x,y) là tọa độ của trạng thái kết thúc.

#### Thuật toán Hill Climbing:

* Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
* Close: tập các trạng thái đã được xét đến, ban đầu close rỗng.

**Bước 1.** Thêm phần trạng thái bắt đầu vào Open.

**Bước 2:** Thực hiện vòng lặp:

* Loop():
  + Kiểm tra nếu Open rỗng => thông báo tìm kiếm thất bại.
  + Gọi O là phần tử đầu của Open, lấy O ở đầu Open.
  + Thêm O vào Close.
  + Nếu O là trạng thái đích => thông báo và kết thúc.
  + Duyệt các trạng thái con do O sinh ra:

o Nếu khác NULL và chưa xuất hiện trong Open và Close thì thêm vào Open sao cho các phần tử được đánh giá giảm dần theo hàm h (hàm h tính bằng số ô khác với trạng thái đích).

# CHƯƠNG 4. THỰC NGHIỆM VÀ ĐÁNH GIÁ

#### Mô tả dữ liệu và thiết lập tham số:

# BẢNG PHÂN CÔNG CÔNG VIỆC

|  |  |  |
| --- | --- | --- |
| **Danh sách thành viên** | **Công việc thực hiện được** | **Tỉ lệ hoàn thành** |

***Bảng 2.*** *Bảng phân công nhiệm vụ*

# TÀI LIỆU THAM KHẢO

[1] [http://ccs.hnue.edu.vn/hoanpt/AI/k60/8-Puzzle-Tri%20tue%20nhan%20tao-](http://ccs.hnue.edu.vn/hoanpt/AI/k60/8-Puzzle-Tri%20tue%20nhan%20tao-K60BC.pdf) K60BC.pdf, mục giới thiệu bài toán.

[2]. [https://en.wikipedia.org/wiki/15\_puzzle,](https://en.wikipedia.org/wiki/15_puzzle) mục giới thiệu trò chơi 15 puzzle

[3].

[https://www.academia.edu/10292262/Ch%C6%B0%C6%A1ng\_4\_T%C3%ACm\_ki%](https://www.academia.edu/10292262/Ch%C6%B0%C6%A1ng_4_T%C3%ACm_ki%E1%BA%BFm_Heuristic)

E1%BA%BFm\_Heuristic, mục cài đặt hàm heuristic

[4].

[https://vi.wikipedia.org/wiki/T%C3%ACm\_ki%E1%BA%BFm\_theo\_chi%E1%BB%8](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng) 1u\_r%E1%BB%99ng, mục giới thiệu thuật toán BFS.

[5]. [https://thanhthao94blog.wordpress.com/2016/08/07/giai-thuat-leo-doi-hill-](https://thanhthao94blog.wordpress.com/2016/08/07/giai-thuat-leo-doi-hill-climbing/) climbing/, mục các vấn đề của thuật toán.

[6]. Giáo trình môn học, Russell, S. J., & Norvig, P. (2016), Artificial Intelligence: A Modern Approach (3rd ed.). Pearson.

[7]. Giáo trình môn học, Lapan, M. (2020). Deep Reinforcement Learning Hands-On. Packt Publishing Ltd.