ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NGUYỄN CÔNG DŨNG
TRẦN QUỐC KHANG
NGUYỄN TRỌNG SƠN

ỨNG DỤNG THUẬT TOÁN TÌM KIẾM TRONG ĐỒ THỊ TÌM KIẾM ĐƯỜNG ĐI TRONG MÊ CUNG

BÁO CÁO
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
(NGÀNH KỸ THUẬT ĐIỆN TỬ VÀ TIN HỌC)

Hà Nội - 2023

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NGUYỄN CÔNG DŨNG
TRẦN QUỐC KHANG
NGUYỄN TRỌNG SƠN

ỨNG DỤNG THUẬT TOÁN TÌM KIẾM TRONG ĐỒ THỊ TÌM KIẾM ĐƯỜNG ĐI TRONG MÊ CUNG

BÁO CÁO
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
(NGÀNH KỸ THUẬT ĐIỆN TỬ VÀ TIN HỌC)

Giảng viên hướng dẫn: Phạm Huy Thông

Hà Nội - 2023

Mục lục

Danh sách hình vẽ

MỞ ĐẦU

Thuật toán tìm kiếm đường đi trong mê cung là một trong những bài toán cơ bản và thường được sử dụng trong môn cấu trúc dữ liệu và giải thuật. Việc tìm kiếm đường đi trong mê cung là một ví dụ điển hình để giới thiệu các thuật toán tìm kiếm và các khái niệm cơ bản như đồ thị, quy hoạch động và các kỹ thuật tối ưu.

Mô phỏng thuật toán tìm kiếm đường đi trong mê cung giúp sinh viên hiểu và tự tìm hiểu các giải thuật, tạo cảm hứng và khai thác triệt để các kiến thức được học. Không chỉ dừng lại ở các mô tả trên lý thuyết, mô phỏng các thuật toán tìm kiếm đường đi trong mê cung giúp sinh viên biết cách thực hiện áp dụng vào các bài tập trên thực tế.

Với sức mạnh của công nghệ, mô phỏng thuật toán tìm kiếm đường đi trong mê cung càng trở nên dễ dàng hơn bằng việc sử dụng các công cụ và thư viện phần mềm để tạo ra một môi trường mô phỏng và thực hiện việc đánh giá các thuật toán tìm kiếm đường đi. Điều này giúp cho môn học cấu trúc dữ liệu và giải thuật có thể được học một cách trực quan và hiệu quả hơn.

Chương 1 TÌM KIẾM ĐƯỜNG ĐI TRONG MÊ CUNG

1.1 Giới thiệu

Mô phỏng tìm đường đi trong mê cung là một phương pháp để tìm kiếm đường đi từ một điểm bắt đầu tới một điểm kết thúc trong một mê cung. Phương pháp này áp dụng các thuật toán tìm kiếm đường đi, như DFS, BFS, A*,... để tìm ra đường đi ngắn nhất hoặc đường đi tối ưu nhất trong mê cung.

Mô phỏng tìm đường đi trong mê cung thường được sử dụng trong lĩnh vực trò chơi điện tử để tạo ra các trò chơi dạng mê cung và giải quyết bài toán đi qua mê cung. Ngoài ra, nó còn được ứng dụng trong các lĩnh vực như điều khiển tàu thủy, máy tính thông minh,...

Trong quá trình mô phỏng, các tường, các đường đi và các điểm đích sẽ được tạo ra. Sau đó, người dùng sẽ nhập vào điểm bắt đầu và điểm kết thúc, và trải nghiệm quá trình tìm đường đi thông qua các thuật toán tìm kiếm đường đi được tích hợp trong mô phỏng.

Mô phỏng tìm đường đi trong mê cung giúp người dùng hiểu rõ hơn về cách thức hoạt động của các thuật toán tìm kiếm đường đi và cách chúng giải quyết bài toán đi qua mê cung.

1.2 Mô tả

Việc mô phỏng thuật toán tìm kiếm đường đi trong mê cung thường bao gồm các bước sau:

1. Tạo ra môi trường mê cung: Đầu tiên, chúng ta cần tạo ra một môi trường mê cung. Điều này có thể được thực hiện bằng cách sử dụng các thuật toán tạo ra mê cung có sẵn, hoặc có thể sử dụng các công cụ thiết kế môi trường để tự tạo ra một mê cung.

2. Xác định các điểm bắt đầu và kết thúc: Sau khi tạo ra mê cung, chúng ta cần xác định các điểm bắt đầu và kết thúc. Điểm bắt đầu là nơi mà thuật toán bắt đầu tìm

kiếm đường đi, trong khi điểm kết thúc là mục tiêu của chúng ta trong quá trình tìm kiếm đường đi.

3. Chọn thuật toán tìm kiếm đường đi: Nhiều thuật toán khác nhau có thể được sử dụng để tìm kiếm đường đi trong mê cung, bao gồm DFS (đi sâu trước), BFS (đi rộng trước), A* và nhiều thuật toán khác. Tùy thuộc vào mục đích và yêu cầu của ứng dụng, chúng ta nên chọn thuật toán tìm kiếm thích hợp nhất để đạt được kết quả tốt nhất.

4. Thực hiện thuật toán tìm kiếm đường đi: Sau khi chọn thuật toán, chúng ta sẽ thực hiện nó để tìm kiếm đường đi từ điểm bắt đầu đến điểm kết thúc. Trong quá trình này, thuật toán sẽ di chuyển qua từng ô trong mê cung, theo các quy tắc của thuật toán được chọn và đưa ra quyết định để di chuyển sang ô tiếp theo.

5. Hiển thị kết quả: Khi thuật toán tìm kiếm đường đi hoàn tất, chúng ta sẽ hiển thị kết quả tìm được cho người dùng, bao gồm đường đi từ điểm bắt đầu đến điểm kết thúc và độ dài của đường đi.

Các bước này sẽ giúp chúng ta thực hiện việc mô phỏng thuật toán tìm kiếm đường đi trong mê cung một cách đơn giản và dễ hiểu. Việc mô phỏng này giúp người dùng hiểu rõ hơn về nguyên lý hoạt động của các thuật toán tìm kiếm đường đi và cách kết hợp chúng để tìm kiếm đường đi tối ưu trong một mê cung.

Chương 2 THUẬT TOÁN TÌM KIẾM

Thuật toán tìm kiếm theo chiều sâu và chiều rộng là hai thuật toán tìm kiếm mù phổ biến, thường được sử dụng trong lý thuyết đồ thị. Chúng ta sẽ đi vào từng thuật toán từ tư tưởng của thuật toán cho tới giả mã và bước đi trong thuật toán để làm rõ hơn cách thức hoạt động của thuật toán. Trong quá trình tìm hiểu ta sẽ nhận thấy chúng có nhiều điểm tương đồng trong cách thực hiện, nhưng cách tổ chức thì khác nhau. Với mỗi thuật toán ta sẽ đưa ra ưu nhược điểm của chúng để có thể sử dụng chúng phù hợp hơn theo những yêu cầu riêng của bài toán đầu vào.

2.1 Thuật toán tìm kiếm theo chiều sâu - Depth First Search (DFS)

Trước tiên ta sẽ đi vào tìm hiểu về thuật toán tìm kiếm theo chiều sâu (DFS) [1]. Thuật toán DFS là một quá trình duyệt hay tìm kiếm trên một cây hoặc một đồ thị. Thuật toán DFS sẽ bắt đầu với một đỉnh gốc và phát triển sâu và xa nhất có thể của mỗi nhánh. Để hiểu hơn ta sẽ đi vào từng phần trong thuật toán:

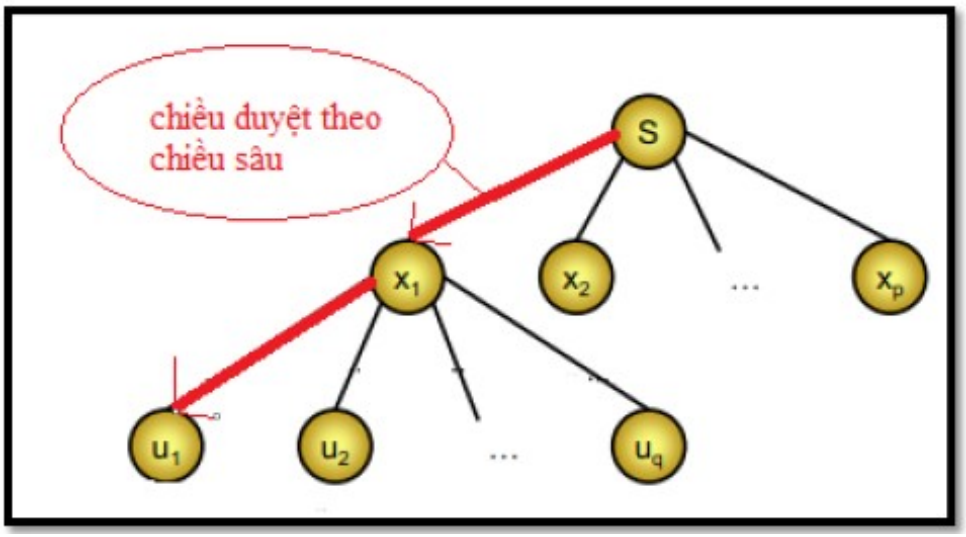
2.1.1 Tư tưởng của thuật toán

Với tư tưởng đi sâu vào từng nhánh, ta giả sử đầu vào của thuật toán là một đồ thị G = (V,E). Coi s là đỉnh gốc của V, ta sẽ bắt đầu quá trình duyệt với s.Từ s ta sẽ đi thăm tới đỉnh kề với s (giả sử ở đây là u0), từ u0 ta tiếp tục quá trình duyệt với các đỉnh kề u0 (trừ các đỉnh đã thăm). Quá trình sẽ tiếp tục tới khi gặp đỉnh cần tìm hoặc đi hết nhánh thì thực hiện lùi lại đỉnh trước đó.

Xét một cách tổng quát thì khi xét một đỉnh u0 ta sẽ có hai khả năng xảy ra:

- Nếu như tồn tại đỉnh v0 kề với u0 mà chưa được thăm thì đỉnh v0 đó sẽ được đánh dấu để trở thành đỉnh đã thăm và quá trình tìm kiếm sẽ bắt đầu từ đỉnh v0 đó.
- Ngược lại, nếu mọi đỉnh kề với u0 đều đã thăm thì ta sẽ quay lại đỉnh mà trước đó ta đến đỉnh u0 để tiếp tục quá trình tìm kiếm

Như vậy trong quá trình thăm đỉnh bằng thuật toán tìm kiếm theo chiều sâu, ta nhận thấy ngay đỉnh càng thăm muộn thì sớm được duyệt trước (đây là cơ chế Last In First Out) . Do đó ta có thể sử dụng thủ tục đệ quy hoặc sử dụng một danh sách kiểu ngăn xếp để tổ chức cho quá trình duyệt của thuật toán. Dưới dây là minh họa cho quá trình duyệt với một cây:



Hình 2.1: Thứ tự duyệt của thuật toán DFS

Hình trên minh họa thứ tự duyệt của thuật toán tìm kiếm theo chiều sâu: ta nhận thấy từ S sẽ thăm X1, tiếp tục tới u1 là kề với X1. Nếu u1 không phải đỉnh tìm và hết nhánh thì sẽ lùi về X1 để thăm u2. Do đó quá trình duyệt sẽ là: S $\rightarrow$ x1 $\rightarrow$ u1 $\rightarrow$ u2 $\rightarrow$ ... $\rightarrow$ uq $\rightarrow$ x2 $\rightarrow$ ...

2.1.2 Giải thuật của thuật toán

Xác định bài toán ta cần lấy ra input và output của bài toán như sau:

- **Input:** đồ thị vào G=(V,E) với đỉnh gốc là s0(Trạng thái đầu)

  Tập đích Goal

- **Output:** một đường đi p từ s đến một đỉnh ftrong tập đích Goals

Thuật toán DFS có 2 cách để duyệt những đỉnh trong quá trình tìm kiếm đó là sử dụng thủ tục đệ quy hoặc sử dụng ngăn xếp để lưu trữ các đỉnh sẽ duyệt tiếp đó. Ta sẽ đi vào

cách sử dụng ngăn xếp để lưu trữ các đỉnh.Ta có các bước cho quá trình thực hiện thuật toán như sau:

Bước 1: khởi tạo

- Các đỉnh đều ở trạng thái chưa đánh dấu, trừ đỉnh xuất phát s là đã đánh dấu.
- Một ngăn xếp S (Stack)ban đầu chỉ đưa vào có một phần tử là s. Bằng việc sử dụng ngăn xếp lưu các đỉnh ta sẽ duyệt sâu vào từng nhánh của đồ thị.

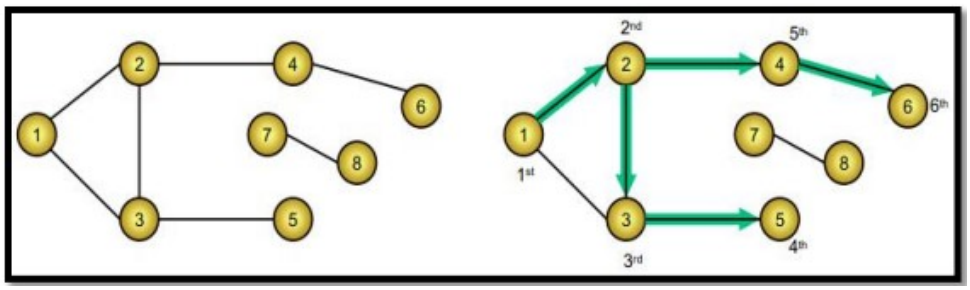
Bước 2: Lặp lại các bước sau cho đến khi ngăn xếp rỗng:

- Nếu ngăn xếp rỗng, không thấy đỉnh đích, thông báo *"không tìm thấy"*, dừng.
- Ngăn xếp không rỗng, lấy u ra khỏi ngăn xếp, thông báo thăm u (bắt đầu duyệt đỉnh u, nếu lần đầu thì là u chính là s).
- Kiểm tra u có phải đỉnh đích t không:
  - Nếu đúng trả về u, dừng vòng lặp, chuyển sang bước 3.
  - Nếu không đúng thì tiếp tục duyệt.
- Xét tất cả các đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
  - Đánh dấu v
  - Ghi nhận đường đi từ v đến u
  - Đẩy v vào ngăn xếp (v sẽ chờ được duyệt tại những bước sau)

Bước 3: Truy ngược lại đường đi (nếu có)

2.1.3 Nhận xét

- Có thể có nhiều đường đi từ s $\rightarrow$ f, nhưng thuật toán DFS luôn trả về một đường đi có thứ tự từ điển nhỏ nhất.
- Quá trình tìm kiếm theo chiều sâu cho ta một cây DFS gốc s. Quan hệ chacon trên cây được định nghĩa là: nếu từ đỉnh u tới thăm đỉnh v thì u là nút cha của nút v. Hình dưới sẽ minh họa cho cây DFS tương ứng với đỉnh xuất phát s=1



Hình 2.2: Cây DFS với s = 1.

Ưu điểm

- Nếu bài toán có lời giải, phương pháp tìm kiếm theo chiều sâu đảm bảo tìm ra lời giải.
- Ký thuật tìm kiếm sâu tập trung vào đích, con người cảm thấy hài lòng khi các câu hỏi tập trung vào vấn đề chính.
- Do cách tìm của kỹ thuật này, nếu lời giải ở rất sâu, kỹ thuật sâu sẽ tiết kiệm thời gian

Nhược điểm

- Tìm sâu khai thác không gian bài toán để tìm lời giải theo thuật toán đơn giản một cách cứng nhắc. Trong quá trình tìm nó không có thông tin nào để phát hiện lời giải. Nếu cọn nút ban đầu không thích hợp có thể không dẫn tới đích của bài toán.
- Không phù hợp với không gian bài toán lớn, kỹ thuật tìm kiếm sâu có thể không đi đến lời giải trong khoảng thời gian vừa phải (nếu cố định thời gian).

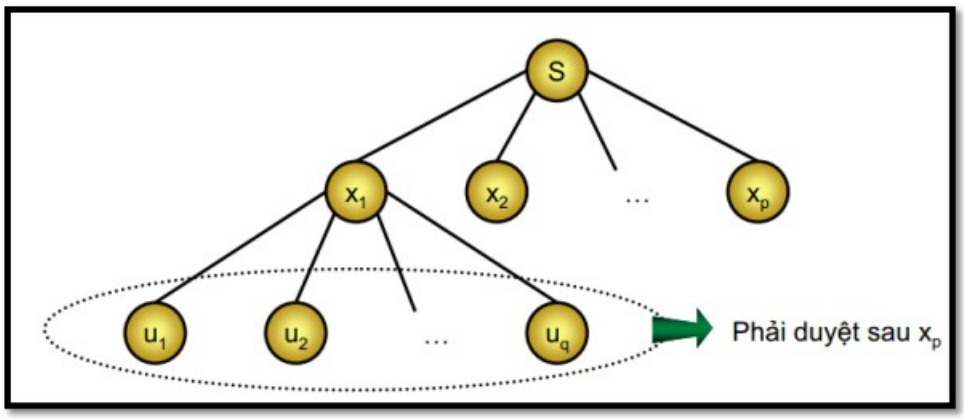
2.2 Thuật toán tìm kiếm theo chiều rộng (Breadth First Sreach)

Tương tự thuật toán tìm kiếm DFS thì thuật toán BFS cũng là một thuật toán phổ biến trong việc tìm kiếm trong đồ thị [2]. Nhưng có đôi chút khác biệt về cách tổ chức các đỉnh để duyệt so với thuật toán DFS. Do đó cách duyệt của BFS cũng trở nên khác DFS. Để thấy sự khác nhau này ta sẽ đi vào tìm hiểu hơn vào thuật toán.

2.2.1 Tư tưởng của thuật toán

Ta giả sử đầu vào thuật toán là một đồ thị G = (V, E), ta sẽ phải thực hiện lập lịch duyệt cho các đỉnh của đồ thị G. Việc duyệt các đỉnh sẽ được ưu tiên sao cho đỉnh nào gần với nó nhất sẽ được duyệt trước. Tức là nó bắt đầu từ mức thấp nhất của không gian bài toán, sẽ duyệt theo chiều từ trái sang phải hoặc ngược lại ở mức tiếp theo, nếu không thấy lời giải ở mức này nó sẽ chuyển xuống mức kế để tiếp tục...cứ như vậy đến khi tìm được lời giải (nếu có). Ta xét ví dụ sau:

Ví dụ : Bắt đầu ta thăm đỉnh S. Việc thăm đỉnh S sẽ phát sinh thứ tự duyệt những đỉnh (x[1], x[2], .. ,x[p]) kề với S (những đỉnh gần S nhất). Khi thăm đỉnh x[1] sẽ lại phát sinh yêu cầu duyệt những đỉnh (u[1], u[2], ..., u[q]) kề với x[1]. Nhưng rõ ràng các đỉnh u này xa S hơn những đỉnh x nên chúng chỉ được duyệt khi tất cả các đỉnh x đã được duyệt xong. Tức là thứ tự duyệt đỉnh sau khi đã thăm x[1] sẽ là : (x[2], x[3], ..., x[p], u[1], u[2],..., u[q])



Hình 2.3: Thứ tự duyệt thuật toán BFS

2.2.2 Giải thuật của thuật toán

- **Input:** cây đồ thị G= (V,E) với đỉnh gốc là s_0 (trạng thái đầu)

  Tập đích Goals.

- **Output:** một đường đi p từ n_0 đến 1 đỉnh f trong tập Goals

Thuật toán sử dụng một cấu trúc dữ liệu là hàng đợi (Queue) để lưu trữ thông tin trung gian trong quá trình tìm kiếm (ở đây dễ hiểu là các đỉnh kế tiếp đợi được duyệt). Tương

tự với tìm kiếm theo chiều sâu, các bước cho giải thuật tìm kiếm theo chiều rộng như sau :

Bước 1: Khởi tạo

- Các đỉnh đều ở trạng thái chưa đánh dấu, trừ đỉnh xuất phát s là đã đánh dấu.
- Một hàng đợi Q (tổ chức dạng hàng đợi Queue), ban đầu chỉ có một phần tử là s. Hàng đợi dùng để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng.

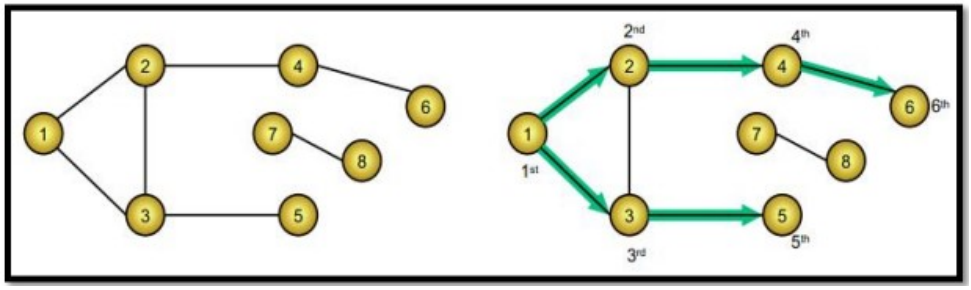
Bước 2: Lặp lại các bước sau cho đến khi hàng đợi rỗng

- Nếu hàng đợi rỗng, không thấy đỉnh đích, thông báo *" không tìm thấy"* , dừng.
- Hàng đợi không rỗng, lấy u ra khỏi hàng đợi , thông báo thăm u (bắt đầu duyệt đỉnh u, nếu là lần duyệt đầu thì u ở đây là s).
- Kiếm tra u có phải đỉnh đích t không
  - Nếu đúng trả về u, dừng vòng lặp, sang bước 3
  - Nếu sai tiếp tục chương trình.
- Xét tất cả các đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
  - Đánh dấu v
  - Ghi nhận đường đi từ v đến u
  - Đẩy v vào hàng đợi (v sẽ chờ được duyệt tại những bước sau)

Bước 3: Truy ngược đường đi

2.2.3 Nhận xét

- Có thể có nhiều đường đi từ s tới f nhưng thuật toán BFS luôn trả về một đường đi ngắn nhất (theo nghĩa đi qua ít cạnh nhất).
- Quá tình tìm kiếm theo chiều rộng cho ta một cây BFS gốc s. Quan hệ cha – con trên cây được định nghĩa là : nếu từ đỉnh u tới thăm đỉnh v thì u là nút cha của nút v. Hình biểu diễn về cây BFS:



Hình 2.4: Biểu diễn cây BFS

Ưu điểm

- Kỹ thuật tìm kiếm rộng là kỹ thuật vét cạn không gian trạng thái bài toán vì vậy sẽ tìm được lời giải nếu có
- Đường đi tìm được thỏa mãn đi qua ít đỉnh nhất.

Nhược điểm

- Tìm kiếm lời giải theo thuật toán đã định trước, do vậy tìm kiếm một cách máy móc; khi không có thông tin bổ trợ cho quá trình tìm kiếm, không nhận ra ngay lời giải.
- Không phù hợp với không gian bài toán có kích thước lớn. Đối với loại bài toán này thì phương pháp tìm kiếm chiều rộng đối diện với các khó khăn về nhu cầu:

  - Cần nhiều bộ nhớ theo số nút cần lưu trữ.

  - Cần nhiều công sức xử lý các nút, nhất là khi các nhánh cây dài, số nút tăng.

  - Dễ thực hiện các thao tác không thích hợp , thừa, đưa đến việc tăng đáng kể số nút phải xử lý.
- Không hiệu quả nếu lời giải ở sâu. Phương pháp này không phù hợp cho trường hợp có nhiều đường dẫn đến kết quả nhưng đều sâu.
- Giao tiếp với người dùng không thân thiện. Do duyệt qua tất cả các nút, việc tìm kiếm không tập trung vào một chủ đề.

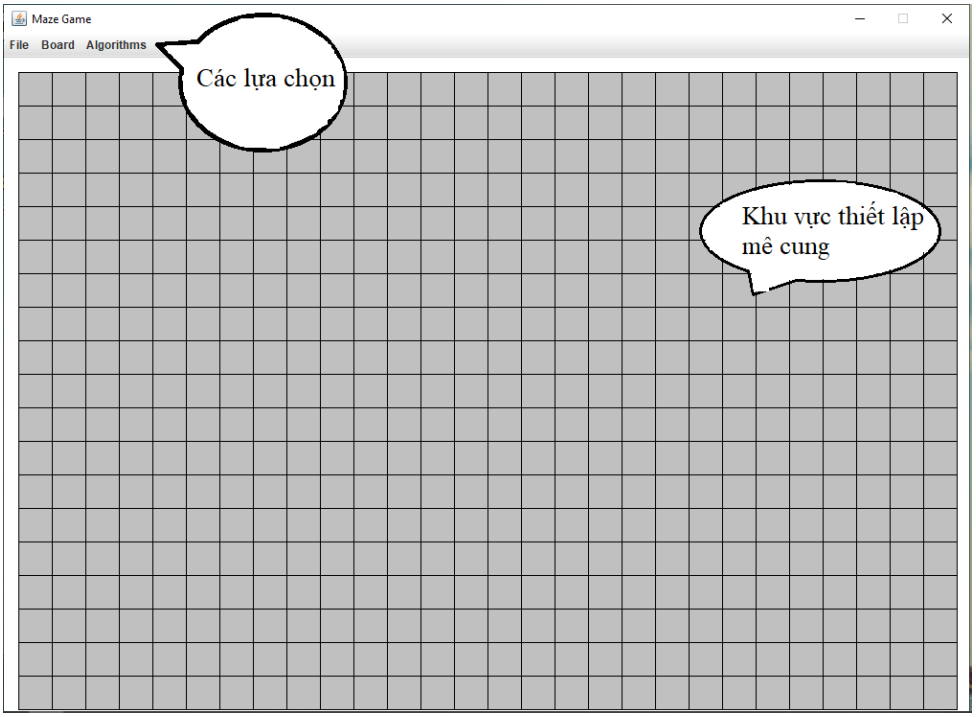
2.3 Độ phức tạp của BFS và DFS

Quá trình tìm kiếm trên đồ thị bắt đầu từ một đỉnh có thể thăm tất cả các đỉnh còn lại, khi đó cách biểu diễn đồ thị có ảnh hưởng lớn tới chi phí về thời gian thực hiện giải thuật:

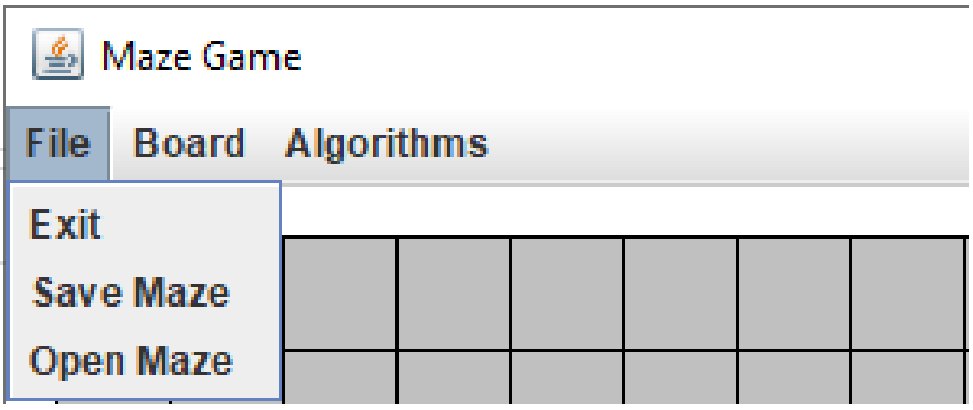
- Trong các trường hợp ta biểu diễn đồ thị bằng danh sách kề, cả hai thuật toán BFS và DFS đều có độ phức tạp tính toán là O(n+m) = O(max(n,m)). Đây là cách cài đặt tốt nhất.
- Nếu ta biểu diễn đồ thị bằng ma trận kề thì độ phức tạp tính toán trong trường hợp này là $O(n+n^2) = O(n^2)$.
- Nếu ta biểu diễn đồ thị bằng danh sách cạnh, thao tác duyệt những đỉnh kề với đỉnh u sẽ dẫn tới việc duyệt toàn bộ danh sách cạnh, đây là cách cài đặt tồi tệ nhất, nó có độ phức tạp tính toán là O(n.m).

Chương 3 KẾT QUẢ

3.1 Giao diện



Hình 3.1: Giao diện game với phần tạo mê cung và các lựa chọn

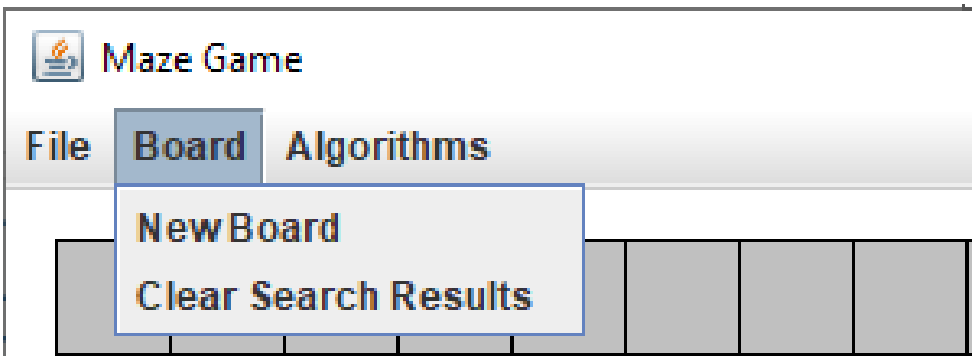


Hình 3.2: Chức năng lựa chọn File

Chức năng lựa chọn **File**:

- Exit: Thoát khỏi trò chơi
- Save Maze: Lưu lại mê cung đã tạo vào máy

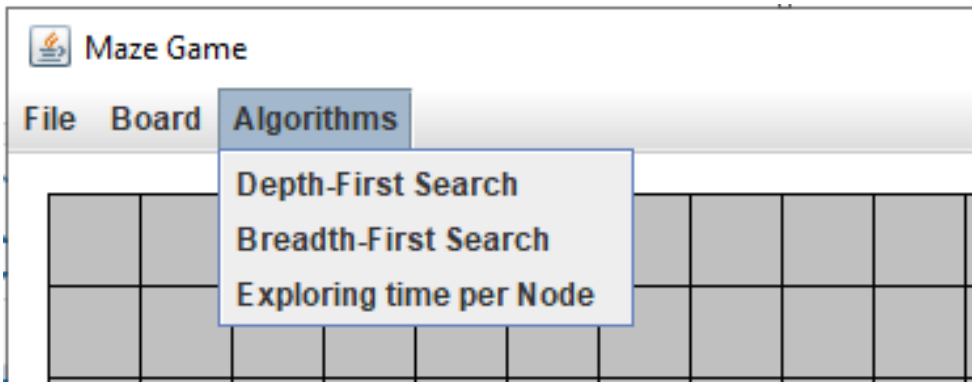
- Open Maze: Mở một mê cung đã có sẵn.



Hình 3.3: Chức năng lựa chọn Board:

Chức năng lựa chọn **Board**:

- New Board: Tạo một giao diện trò chơi mới
- Clear Search Result: Xóa đường đi thuật toán đã tìm kiếm

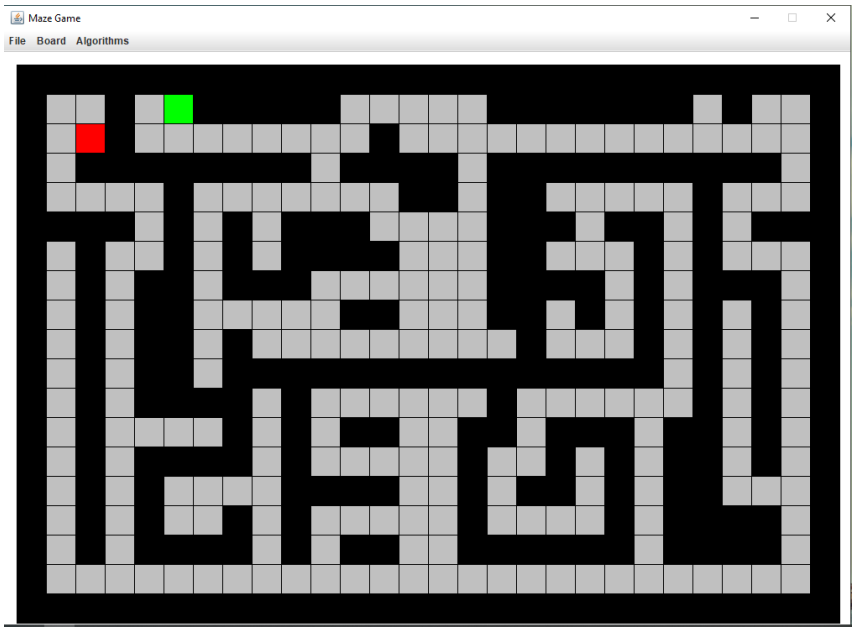


Hình 3.4: Chức năng lựa chọn Algorithms

Chức năng lựa chọn **Algorithms**:

- Depth-First Search: Sử dụng thuật toán DFS
- Breadth-First Search: Sử dụng thuật toán BFS
- Exploring time per Node: Cài đặt thời gian

Giao diện mê cung đã được thiết lập với các thông tin:



Hình 3.5: Giao diện mê cung đã được thiết lập với các thông tin

- Ô màu đen: Tường, chướng ngại vật không thể di chuyển qua được
- Ô màu xám: Vị trí có thể đi qua
- Ô màu đỏ: Vị trí đích
- Ô màu xanh: Vị trí bắt đầu

3.2 Kết quả

3.2.1 Thuật toán DFS

```
public void dfs(Node start) {
        Stack<Node> nodes = new Stack<>();
        nodes.push(start);

        while (!nodes.empty()) {

                Node curNode = nodes.pop();
                if (!curNode.isEnd()) {

                        curNode.setColor(Color.ORANGE);
                        try {
                                Thread.sleep(searchtime);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        curNode.setColor(Color.BLUE);
                        for (Node adjacent : curNode.getNeighbours()) {
                                nodes.push(adjacent);
                        }

                } else {
                        curNode.setColor(Color.MAGENTA);
                        break;
                }
        }

}
```

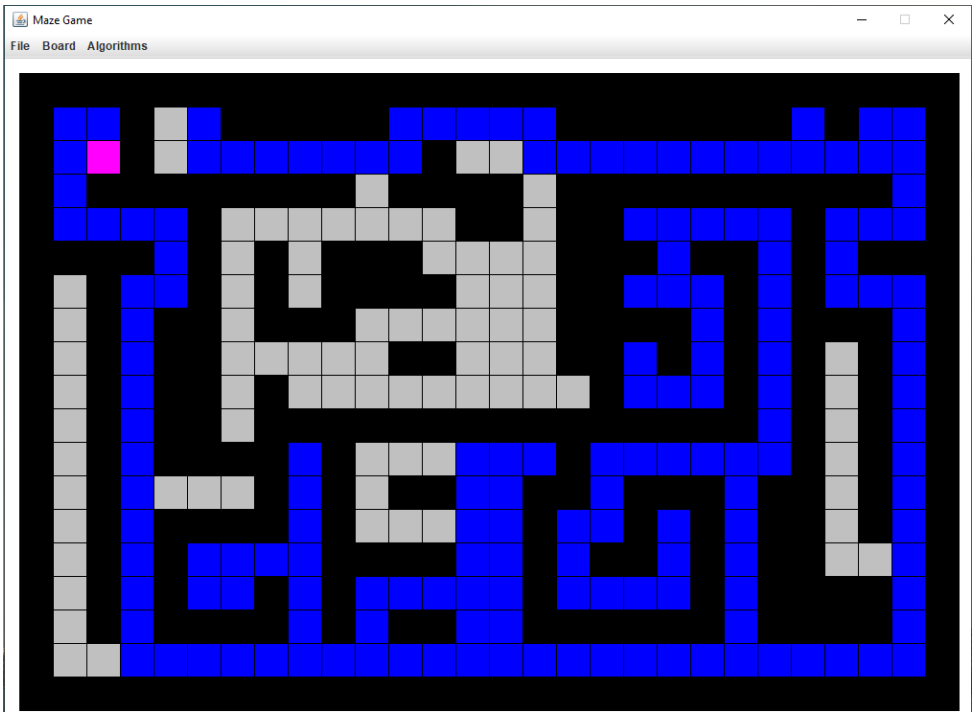
Hình 3.6: Thuật toán DFS tìm kiếm đường đi theo chiều sâu

Trong đó:

1. Tạo một đối tượng ’Stack<Node>’ để lưu trữ các nút chưa được duyệt trong quá trình tìm kiếm.
2. Đưa nút ’start’ vào ’Stack’ và bắt đầu vòng lặp while khi ’Stack’ chưa rỗng.
3. Lấy ra nút đầu tiên trong ’Stack’ dùng ’pop()’
4. Kiểm tra nếu nút hiện tại là nút đích (’isEnd()’) thì đổi màu thành màu Magenta và dừng thuật toán.
5. Nếu nút không phải đích, đổi màu thành màu cam và ngủ trong một khoảng thời gian (tùy chọn).
6. Đổi màu của nút về màu xanh dương và thêm các nút kề vào ’Stack’.

7. Lặp lại từ bước 3.

8. Khi thuật toán kết thúc, đường đi từ nút bắt đầu đến nút đích sẽ được tô màu Magenta.



Hình 3.7: Kết quả tìm kiếm đường đi ma trận bằng thuật toán DFS

3.2.2 *Thuật toán BFS*

```
public void bfs(Node start, Node end, int graphWidth, int graphHeight) {
        Queue<Node> queue = new LinkedList<>();
        Node[][] prev = new Node[graphWidth][graphHeight];

        queue.add(start);
        while (!queue.isEmpty()) {

                Node curNode = queue.poll();
                if (curNode.isEnd()) {
                        curNode.setColor(Color.MAGENTA);
                        break;
                }

                if (!curNode.isSearched()) {
                        curNode.setColor(Color.ORANGE);
                        try {
                                Thread.sleep(searchtime);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        curNode.setColor(Color.BLUE);
                        for (Node adjacent : curNode.getNeighbours()) {
                                queue.add(adjacent);
                                prev[adjacent.getX()][adjacent.getY()] = curNode;


                        }
                }
        }

}
```

Hình 3.8: Thuật toán BFS tìm kiếm đường đi theo chiều rộng

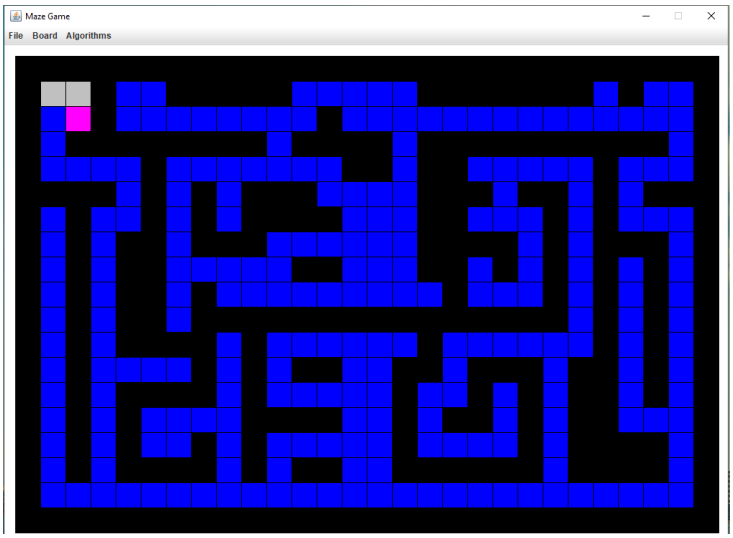
Trong đó:

1. Tạo một ’Queue<Node>’ để lưu trữ các nút chưa được duyệt.
2. Khởi tạo một mảng 2 chiều ’Node[][]’ để lưu trữ nút đi trước của nút hiện tại.
3. Thêm nút ’start’ vào hàng đợi (’queue’) và bắt đầu vòng lặp while khi hàng đợi chưa rỗng.
4. Lấy nút đầu tiên ra khỏi hàng đợi dùng ’poll()’.
5. Nếu nút là nút đích (’isEnd()’), thì đổi màu của nút thành Magenta, và kết thúc thuật toán.
6. Nếu nút chưa được tìm kiếm, thì đổi màu thành Orange và ngủ một khoảng thời gian tùy chọn.
7. Đổi màu của nút về màu Blue và thêm các nút kề của nó vào hàng đợi (’queue’).

8. Lưu trữ nút đóng vai trò là nút đi trước của nút hợp lệ trong mảng prev dựa trên vị trí hàng và cột của nút trong đồ thị.

9. Lặp lại từ bước 4.

10. Khi thuật toán kết thúc, tìm đường đi ngắn nhất từ nút ’start’ đến nút ’end’ sử dụng mảng ’prev’ và sắp xếp các nút trên đường đi này thành màu Magenta.



Hình 3.9: Kết quả tìm kiếm đường đi ma trận bằng thuật toán BFS

KẾT LUẬN

Thông qua việc tìm hiểu và nghiên cứu về đề tài này giúp chúng em có cái nhìn toàn diện hơn trong việc ứng dụng thuật toán tìm kiếm trong đồ thị tìm kiếm đường đi trong mê cung. Bài toán tìm kiếm đường đi trong mê cung là chủ đề đã được nhiều người nghiên cứu giải quyết nhưng cho đến nay vẫn chưa có cách giải quyết tối ưu cho tất cả trạng thái. Hy vọng việc áp dụng thuật toán tìm kiếm trong đồ thị sẽ góp phần bổ sung thêm một hướng giải quyết cho bài toán. Do thời gian có hạn nên đề tài không tránh khỏi sai sót, mong thầy góp ý, đánh giá giúp chúng tôi hoàn thiện đề tài

Tài liệu tham khảo

[1] Wikipedia. “Tìm kiếm theo chiều sâu”. In: (2023). URL: https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_s%C3%A2u.

[2] Wikipedia. “Tìm kiếm theo chiều rộng”. In: (2023). URL: https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng.