**1. Giải thích Chi tiết về Thuật toán**

**1.1 Thuật toán Uniform Cost Search (UCS)**

- **Mục tiêu**: UCS tìm con đường ít tốn kém nhất trong mê cung, trong đó mỗi bước di chuyển của Ares hoặc đẩy đá đều có chi phí khác nhau.

- **Nguyên lý hoạt động**:

  - Chương trình khởi tạo hàng đợi ưu tiên (min-heap) với trạng thái bắt đầu, bao gồm vị trí ban đầu của Ares, vị trí của các viên đá và tổng chi phí khởi điểm là 0.

  - Hàng đợi ưu tiên đảm bảo rằng thuật toán sẽ khám phá các con đường có chi phí thấp nhất trước, giúp UCS đạt được tính tối ưu khi tìm được trạng thái đích với nỗ lực thấp nhất.

  - **Điều kiện dừng**: UCS sẽ dừng lại khi đạt được trạng thái mà tất cả các viên đá đều nằm trên các công tắc.

- **Tính toán Chi phí**:

  - Di chuyển bình thường có chi phí là 1, khi đẩy viên đá sang một ô liền kề, chi phí sẽ được cộng thêm bằng trọng lượng của viên đá. Chi phí đẩy này là tổng cộng và giúp UCS ưu tiên các con đường ít phải đẩy đá.

**2.2 Cấu trúc Dữ liệu và Các Hàm**

- **Hàng đợi ưu tiên**: Lưu trữ các trạng thái theo thứ tự chi phí tăng dần, giúp UCS mở rộng con đường có chi phí thấp nhất trước.

- **Tập trạng thái đã được duyệt:** Lưu trữ các trạng thái đã được duyệt, bao gồm vị trí của Ares và các vị trí của tất cả viên đá. Điều này giúp UCS tránh duyệt lại các trạng thái, tối ưu hóa bộ nhớ và giảm các tính toán dư thừa.

**2.3 Lưu đồ Hoạt động**

1. **Khởi tạo Trạng thái:** Chương trình đọc cấu trúc mê cung, xác định vị trí của Ares, các viên đá và công tắc, đồng thời khởi tạo trọng lượng của các viên đá.

2. **Thực thi UCS**:

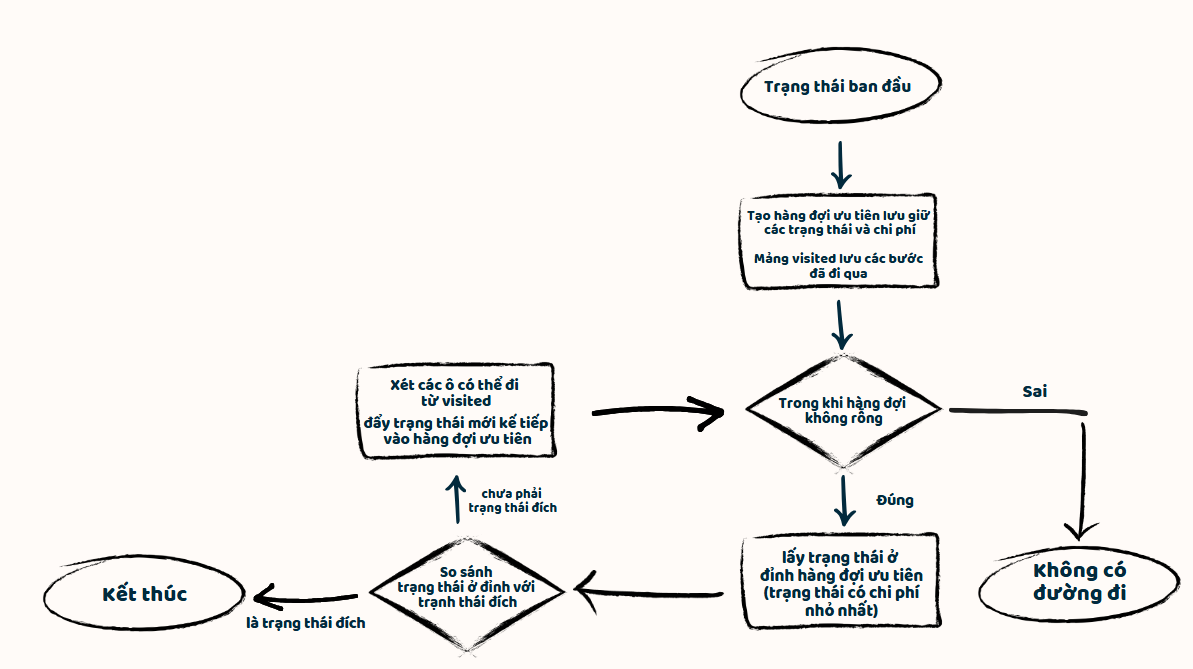
   - Lấy trạng thái có chi phí thấp nhất khỏi hàng đợi ưu tiên.

   - Với mỗi hành động di chuyển hoặc đẩy đá khả thi, tính toán trạng thái và chi phí mới.

   - Thêm các trạng thái hợp lệ vào hàng đợi ưu tiên nếu chưa được duyệt, ưu tiên các con đường rẻ nhất.

   - Lặp lại cho đến khi đạt được trạng thái đích, trả về con đường và các số liệu thống kê.

**3. Minh họa**

Biểu đồ khối của thuật toán được sử dụng như sau:

A diagram of a game

Description automatically generatedXem xét cấu trúc mê cung đơn giản sau, giả sử viên đá có khối lượng là 3:

Sau khi xét các trạng thái có thể đi tiếp theo và chi phí của nó, bây giờ trong hàng đợi ưu tiên có các trạng thái là: **pq[(1, [‘l’]), (1, [‘r’]), (4, [‘U’])]**. Ở lần lặp tiếp theo, trạng thái được lấy ra là trạng thái có chi phí là nhỏ nhất: **(1, [‘l’])** và đồng thời ô tại vị trí **[5, 3]** cũng được thêm vào tập hợp **visited**.

A diagram of a game

Description automatically generatedỞ lần lặp có trạng thái Ares ở ô **[5, 2]**, ta chi tiếp tục có 2 trạng thái kế tiếp:

Lúc này nếu chú ý ta thấy khi một trạng thái lấy từ hàng đợi ưu tiên thì vị trí của trạng thái này không thể đi qua được nữa, tránh việc duyệt lại trạng thái cũ. Và tương tự chúng ta lại thêm hai trạng thái tiếp theo vào hàng đợi ưu tiên, hàng đợi lúc này trở thành:  
**pq[(1, [‘r’]), (4, [‘U’]), (2, [‘l, u’]), (2, [l, l])]**.

Lúc này ta lại tiếp tục thực hiện lấy trạng thái có chi phí nhỏ nhất trong đường đi là **(1, [‘r’])** và tiếp tục khám phá ở đường đi này.

Thoạt nhìn, ta thấy đường đi ngắn nhất cho Ares là đẩy thẳng viên đá lên trên 2 ô để tới đích, nhưng vì thuật toán UCS là một thuật toán tìm kiếm mù (không có thông tin về kết quả cần đạt) nên Ares sẽ khám phá các khả năng đi dựa trên chi phí nhỏ nhất mà các hướng đi có.

Mặc dù ban đầu dường như Ares di chuyển sang 2 bên là vô ích, nhưng sau khi di chuyển tới một mức độ thì chi phí cho đường đi của Ares sẽ lớn hơn chi phí ban đầu Ares đẩy viên đá lên trên **(1, [‘U’])** nên Ares sẽ chọn hướng đẩy viên đá lên trên. Và cứ thế tiếp tục Ares sẽ xét các nhánh trạng thái khác chi phí nhỏ hơn nhưng chung quy các khả năng này đều không tới được đích với chi phí nhỏ hơn chi phí của việc đẩy viên đá thêm một lần lên trên để chạm đích **(8, [‘U’, ‘U’])**.

**5. Thách Thức**

- **Tiêu thụ bộ nhớ**: UCS có thể tiêu tốn nhiều bộ nhớ trong các mê cung lớn vì phải theo dõi nhiều lộ trình và giữ chúng trong bộ nhớ cho đến khi đạt được trạng thái đích. Trường hợp thử nghiệm lớn nhất cho thấy mức tiêu thụ bộ nhớ cao (248.72 MB).

- **Mở rộng nút:** Vì UCS mở rộng tất cả các con đường khả thi để duy trì tính tối ưu, số lượng nút được tạo có thể lớn, đặc biệt trong các mê cung có nhiều viên đá và bố trí phức tạp.

- **Thời gian thực thi**: Số lượng nút tăng trực tiếp ảnh hưởng đến thời gian thực thi. Trong các mê cung lớn hoặc khi viên đá ở xa công tắc, UCS có thể mất nhiều thời gian để đánh giá tất cả các con đường.

**6. Hiệu suất thuật toán**

- **Tính tối ưu**: UCS đảm bảo tìm ra con đường tối ưu nhất về chi phí do mở rộng các con đường theo thứ tự chi phí tăng dần. Điều này rất có lợi cho các bài toán như đẩy đá vào công tắc trong mê cung, nơi mỗi động tác và đẩy đá đều ảnh hưởng đến tổng chi phí.

- **Thách thức của UCS trong các mê cung lớn**: Khi độ phức tạp của mê cung tăng lên, việc khám phá toàn diện của UCS trở nên tốn kém hơn về bộ nhớ và thời gian. Đối với các mê cung cực kỳ phức tạp, các thuật toán sử dụng heuristic như A\* (với ước lượng chi phí để hoàn thành) có thể mang lại hiệu suất tốt hơn với một chút đánh đổi trong tính tối ưu.

**7. Kết luận**

Chương trình đã giải hiệu quả bài toán mê cung bằng cách sử dụng UCS, đảm bảo con đường tối thiểu chi phí để di chuyển các viên đá vào công tắc. Mặc dù tính toàn diện của UCS cung cấp giải pháp tốt nhất, các mê cung lớn tạo ra thách thức về bộ nhớ và thời gian, đặc biệt khi các viên đá có trọng lượng khác nhau và nhiều công tắc. Chương trình này cung cấp một giải pháp mạnh mẽ nhưng có thể được cải tiến thêm bằng cách bổ sung heuristic cho các phiên bản tương lai, đặc biệt để xử lý các mê cung phức tạp hơn một cách hiệu quả.