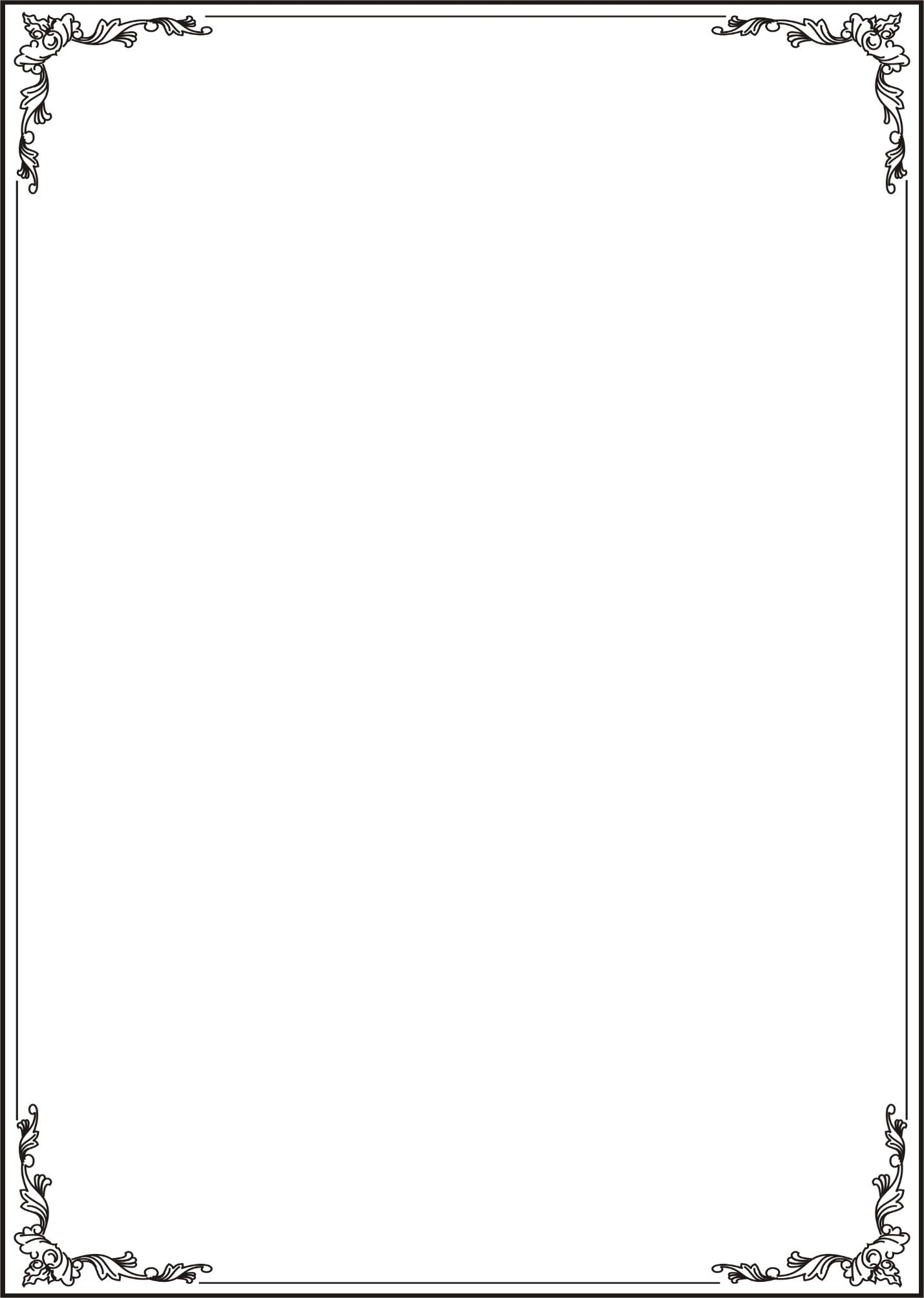
** ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

----🙞🙜🕮🙞🙜----

Logo

Description automatically generated

**TIỂU LUẬN**

**MÔN: CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN**

**Đề Tài:**

**Xây dựng thuật toán tìm đường đi ngắn nhất dựa trên giải thuật A\*.**

Giảng viên hướng dẫn:

Nguyễn Thị Hồng Minh

Phạm Huy Thông

Sinh viên thực hiện 1: Trần Ngọc Hải.

Mã sinh viên: 20001911.

Sinh viên thực hiện 2: Lưu Hiểu Huy.

Mã sinh viên: 20001926.

Lớp: K65A3-MT&KHTT.

Mục Lục.

[**I. Giới thiệu về thuật toán A\*:** 4](#_Toc105357025)

[**II. Mô tả thuật toán:** 4](#_Toc105357026)

[*2.1. Các khái niệm liên quan.* 4](#_Toc105357027)

[**1) Khoảng cách Mahattan:** 5](#_Toc105357028)

[**2) Khoảng cách theo đường chéo:** 6](#_Toc105357029)

[**3) Khoảng cách Euclidean:** 6](#_Toc105357030)

[*2.2. A\* hoạt động như thế nào?* 9](#_Toc105357031)

[**III. Tính đúng đắn của thuật toán.** 11](#_Toc105357032)

[**IV. Triển khai thuật toán.** 13](#_Toc105357033)

[*4.1. Triển khai thuật toán.* 13](#_Toc105357034)

[*4.2. Nhận xét.* 17](#_Toc105357035)

[**V. Ứng dụng của giải thuật.** 19](#_Toc105357036)

[**Kết Thúc.** 20](#_Toc105357037)

# **I. Giới thiệu về thuật toán A\*:**

A\* là một thuật toán tìm kiếm đường đi và duyệt đồ thị, thường được sử dụng trong nhiều lĩnh vực liên quan đến khoa học máy tính nhờ độ tối ưu và sự hiểu quả mà nó mang lại. Thuật toán ban đầu được thiết kế như một bài toán duyệt đồ thị để giúp robot có thể tự tìm đường đi.

Thuật toán được công bố lần đầu tiên vào năm 1968 bởi Peter Hart, Nils Nilsson và Bertram Raphael thuộc viện nghiên cứu Stanford (nay là tổ chức SRI International).

Giải thuật A\* về cơ bản giống với thuật toán Dijkstra nhưng thay vì có thể tìm được đường đi ngắn nhất từ một đỉnh cho trước đến tất cả các đỉnh khác trên đồ thị, nó chỉ có thể tìm được đường đi ngắn nhất giữa hai đỉnh bất kỳ. Tuy vậy, A\* lại đạt được hiệu suất tốt hơn nhiều so với Dijkstra nhờ việc sử dụng hàm heuristic. Hàm heuristic này cũng khiến cho thuật toán trở nên thông minh hơn, có thể nói rằng nhờ có heuristic mà thuật toán có thể lập tức tìm kiếm được đúng hướng để đi đến đích.

# **II. Mô tả thuật toán:**

## *2.1. Các khái niệm liên quan.*

Diagram

Description automatically generatedA\* là một thuật toán tìm kiếm đường đi có hiểu biết từ một node bất kỳ đến một node đích với chi phí đường đi nhỏ nhất. Nó thực hiện điều này bằng cách duy trì một cây các đường đi từ node bắt đầu và kéo dài các đường dẫn đó (mỗi một lần lặp chỉ kéo dài một cạnh) cho đến khi nó đạt được điều kiện dừng của thuật toán.

*(hình ảnh minh họa)*

Tại mỗi vòng lặp, thuật toán sẽ cần xác định xem đỉnh (cạnh) nào sẽ được mở rộng tiếp theo. Để làm được như vậy, nó dựa trên chi phí hiện tại của đường đi và chi phí ước tính để có thể tới được điểm đích. Cụ thể, A\* sẽ chọn đường đi có giá trị hàm *f(n)* nhỏ nhất với:

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

Trong đó:

* n: Là node tiếp theo trên đường đi (đường đi đang được xét).
* g(n): Là chi phí đường đi từ node bắt đầu cho tới node n.
* h(n): Còn gọi là hàm heuristic, nó ước tính chi phí của đường đi từ node n cho tới node đích.

Hàm h(n) có thể có nhiều cách tính, nó phụ thuộc vào tình huống của bài toán. Sau đây là hai trong số các phương pháp ước tính chi phí cho hàm h(n):

### **1) Khoảng cách Mahattan:**

Trong công thức này, h(n) được tính bằng tổng sự chêch lệch tọa độ x và y của node đích so với node đang được xét.

h(n) =

Chúng ta sử dụng khoảng cách Mahattan trong trường hợp chỉ có đường đi theo 4 hướng giữa các node (trái – phải – trên – dưới), thường được sử dụng điển hình trong các trò chơi điện tử.

A picture containing text, shoji, crossword puzzle, building

Description automatically generated

Ở đây ta xuất phát từ điểm màu đỏ (0, 0) đến điểm màu xanh (7, 3). Điểm màu đỏ ở đây sẽ có h(n) = |0-7| + |0-3| = 10.

### **2) Khoảng cách theo đường chéo:**

Tương tự như Mahattan, nó cũng ước lượng hàm h(n) dựa vào sự chênh lệch về tọa độ của điểm hiện tại và điểm đích.

* + - * D là khoảng cách giữa các node theo phương ngang/dọc (thường có giá trị bằng 1, tùy từng tình huống).
      * D2 là khoảng cách theo đường chéo giữa các node (thường bằng căn bậc 2).

Chúng ta sử dụng khoảng cách loại này trong trường hợp các hướng di chuyển có thể khả quan theo 8 hướng (thêm 2 đường chéo so với 4 hướng). Giống như các nước di chuyển của quân Hậu trong cờ vua.

A picture containing shoji, crossword puzzle, building

Description automatically generated

### **3) Khoảng cách Euclidean:**

Đúng như tên gọi của nó, hàm h(n) sẽ tính khoảng cách giữa 2 điểm dựa vào tọa độ của chúng.

h(n) =

Chúng ta sử dụng cách tính này khi ta có thể đi theo bất kỳ hướng nào để có thể di chuyển từ node này sang node khác.

Chart, line chart

Description automatically generated

Đối với mô phỏng dạng mê cung (lưới) thế này và kể cả trong hầu hết các tựa game về bản đồ có sử dụng thuật toán A\*, thì bản chất của nó vẫn là việc duy trì một cây các đường đi từ node bắt đầu tới node đích và *đây cũng là phương pháp mô tả đồ thị của chúng em trong phần demo thuật toán*.

Hình ảnh dưới đây sẽ giúp ta dễ hình dung hơn về cách minh họa này:

Qr code

Description automatically generated

**VS**

Diagram, shape

Description automatically generated

*Rõ ràng ta thấy, ở cả 2 cách mô tả đồ thị, thuật toán vẫn đang duy trì một cây từ điểm đầu tới điểm đích chẳng qua các nhánh đang được xếp chồng lên nhau mà thôi.*

## *2.2. A\* hoạt động như thế nào?*

Để hiểu được rõ hơn thuật toán hoạt động như thế nào, chúng ta sẽ xét qua ví dụ dưới đây:

Diagram, shape

Description automatically generated

Giả sử ta có một cây đơn giản thế này, mục tiêu của ta làm tìm được đường đi ngắn nhất từ đỉnh S đến đỉnh E theo như trên đồ thị. Giá trị hàm h(n) trong những trường hợp như thế này thường được tính theo khoảng cách Euclidean. Ở đây các giá trị hàm h đã được cho trước để ta dễ dàng tính toán.

**Lưu ý:** Các đỉnh (node) có giá trị hàm f nhỏ hơn, sẽ có độ ưu tiên cao hơn cho nên chúng sẽ được duyệt trước.

Lần lặp 1: Mở rộng từ đỉnh S (đỉnh bắt đầu):

{S, A}: f(A) = SA + h(A) = 1 + 5 = 6 (SA = g(A))

{S, B}: f(B) = SB + h(B) = 2 + 6 = 8 (SB = g(B))

Lần lặp 2: Mở rộng từ đỉnh A:

{S, B}: f(B) = 2 + 6 = 8

{S, A, X}: f(X) = (g(A) + AX) + h(X) = (1+4) + 5 = 10 (g(X) = g(A) + AX)

{S, A, Y}: f(Y) = (g(Y) + AY) + h(Y) = (1+7) + 8 = 16 (g(Y) = g(A) + AY)

Lần lặp 3: Mở rộng từ đỉnh B:

{S, A, X}: f(X) = (g(A) + AX) + h(X) = (1+4) + 5 = 10 (g(X) = g(A) + AX)

{S, B, C}: f(C) = (g(B) + BC) + h(C) = (2+7) + 4 = 13 (g(C) = g(B) + BC)

{S, A, Y}: f(X) = (g(A) + AY) + h(Y) = (1+7) + 8 = 16 (g(Y) = g(A) + AY)

{S, B, D}: f(D) = (g(B) + BD) + h(D) = (2+1) + 15 = 18 (g(D) = g(B) + BD)

Lần lặp 4: Mở rộng từ đỉnh X:

{S, A, X, E}: f(E) = (g(X) + XE) + 0 = 5 + 2 = 7 (h(E) = 0 do E là điểm đích)

{S, B, C}: f(C) = (g(B) + BC) + h(C) = (2+7) + 4 = 13

{S, A, Y}: f(Y) = (g(A) + AY) + h(Y) = (1+7) + 8 = 16

{S, B, D}: f(D) = (g(B) + BD) + h(D) = (2+1) + 15 = 18

Đến lần lặp thứ 5, nó nhận ra E là điểm đích, điều kiện dừng được thỏa mãn. Lúc này thuật toán đã tìm ra được đường đi tối ưu để tới điểm đích. Chi phí cho đường đi chính là giá trị g(E).

Qua ví dụ trên, ta cũng thấy rằng A\* hoạt động tốt cho cả trường hợp đồ thị có trọng số.

# **III. Tính đúng đắn của thuật toán.**

Để dễ hình dung hơn về công thức này, chúng ta sẽ phân tích nó như sau:

Chart, scatter chart

Description automatically generated

Trước hết, chúng ta có thể thấy rằng để đi từ node bắt đầu đến node đích, đường đi ngắn nhất chính là đường thẳng nối giữa chúng, ta gọi nó là *đường đi tối ưu nhất* và độ dài của nó là *dmin*. Bất kỳ một node n nào nằm trên đó sẽ đều có f(n) = g(n) + h(n) = dmin. Do đó, từ node bắt đầu, các node xung quanh nó sẽ được duyệt và dần mở rộng ra từ những node có giá trị hàm f(n) nhỏ nhất (= dmin) và cuối cùng đến được node đích.

Vậy tại sao thuật toán lại chọn mở rộng đường đi từ những node có giá trị hàm f nhỏ nhất?

Chart

Description automatically generated

Giả sử có một node m nào đó nằm ngoài đường đi tối ưu nhất trên, tức là ta sẽ có f(m) = g(m) + h(m) > dmin do tính chất của tam giác. Điều này có nghĩa là nếu ta xét node m để mở rộng cây thì khi xây dựng lại đường đi, ta phải đi qua node m, trong khi đó các node có giá trị hàm f = dmin như node n mới cho ta đường đi ngắn nhất.

Trong trường hợp không có đường đi thẳng nào từ node bắt đầu đến node đích, việc chọn một điểm m rất gần với đường đi tối ưu nhất sẽ cho ta một đường đi khác ‘gần tốt bằng’ với đường đi tối ưu nhất, do f(m) lúc này xấp xỉ dmin và đó cũng là sự lựa chọn tốt nhất ta có thể có.

Ta cũng dễ dàng nhận thấy rằng, những node có giá trị hàm f càng lớn sẽ là những node nằm càng xa đường đi tối ưu nhất, điều đó đồng nghĩa với việc đường đi mà ta chọn càng dài, càng ít tối ưu.

Từ đó, ta có thể kết luận rằng việc A\* lựa chọn mở rộng đường đi theo các node có giá trị hàm f nhỏ nhất là rất hợp lý.

Từ đây, chúng ta cũng có thể nhận xét rằng, nếu nói thuật toán này “có hiểu biết” cũng không hề sai. A\* chọn lọc các node cần đi để tới được đích nhờ có heuristic. Đây cũng là lý do vì sao, A\* tốt hơn rất nhiều so với Dijkstra.

# **IV. Triển khai thuật toán.**

## *4.1. Triển khai thuật toán.*

Có nhiều cách khác nhau để triển khai thuật toán A\*, mỗi một cách triển khai cũng ít nhiều ảnh hưởng đến hiệu suất của thuật toán. Vậy nên trong báo cáo này, chúng em sẽ trình bày cách triển khai mà chúng em đã tiến hành trong phần source code java.

Trước hết, bản đồ sẽ được mô phỏng dưới dạng mê cung (lưới), ưu điểm của cách mô phỏng này là dễ thực hiện, có thể sinh mê cung rộng (có cả vật cản) theo ý muốn cùng với biến toàn cục SPEED giúp điều chỉnh tốc độ tìm kiếm của thuật toán sẽ cho ta thấy được hàng đợi ưu tiên hoạt động thế nào, từ đó có những điều chỉnh hợp lý cho thuật toán.

Sau đây là các yếu tố cần thiết để triển khai thuật toán:

1. Một lớp để lưu lại các thông tin, phương thức của ‘đối tượng’ node sẽ và đã được xét tới khi thực hiện thuật toán. Về cơ bản, lớp này gồm các thông tin về tọa độ, biến tham chiếu tới node cha (giúp cho việc xây dựng lại đường đi) hay giá trị hàm g, hàm h của node đó.

2. Một hàng đợi ưu tiên (Priotrity Queue) để lưu các node đang chờ duyệt (ta gọi là OpenList). Hàng đợi ưu tiên này được chúng em xây dựng dựa trên ArrayList, việc xây dựng dựa trên lớp này giúp chúng em tận dụng được những phương thức đã có sẵn, chỉ cần sửa lại một chút cho phù hợp với thuật toán.

Trong OpenList, ta sẽ sắp xếp các node theo thứ tự ưu tiên là giá trị hàm f(n) càng lớn, độ ưu tiên càng cao. Điều này có phần hơi ngược so với lý thuyết bên trên nhưng về bản chất thì nó vẫn giống nhau, việc ta sắp xếp danh sách theo thứ tự giảm dần của hàm f(n) không làm thay đổi ý tưởng thuật toán, vốn dĩ ta phải làm như vậy là khi loại bỏ phần tử đã xét xong ra khỏi OpenList, độ phức tạp của nó sẽ là O(1) thay vì O(n) nếu xếp theo thứ tự tăng dần.

Các phương thức của OpenList như sau:

Phương thức insert của Priority Queue sẽ gọi đến phương thức add được Override lại từ lớp ArrayList.

Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Do danh sách đã được sắp xếp giảm dần, nên phần tử cuối cùng sẽ là phần tử có giá trị hàm f nhỏ nhất.

Graphical user interface, text, application, chat or text message

Description automatically generated

3. Một danh sách lưu các node đã được duyệt (gọi là ClosedList). Nó đơn giản chỉ là một ArrayList lưu các node đã được duyệt.

4. Một lưu ý ở đây đó là điều kiện một node có được thêm vào OpenList để chờ xét hay không? Hai điều kiện đáng lưu ý nhất ở đây đó là: Nếu node đó đã có trong ClosedList hoặc nó đang ở trong OpenList, mà giá trị hàm f của nó ở lần duyệt trước đó, nhỏ hơn giá trị hàm f của nó với lần duyệt hiện tại, thì nó sẽ không được thêm vào OpenList.

5. Một phương pháp ước tính hàm h(n). Ở đây chúng em sử dụng phương pháp tính hàm h(n) theo khoảng cách Mahattan, bởi về trực quan thì đường đi phải đủ rộng mới được tính là có lối đi (thuật toán vẫn hoạt động tốt với cả phương chéo).

Sau đây là phần cài đặt chi tiết cho thuật toán:

Text

Description automatically generated with medium confidence

Về cơ bản, thuật toán sẽ tiếp tục tìm kiếm khi mà trong OpenList còn phần tử (tức là còn có node để mở rộng).

Từ OpenList, lấy ra phần tử có giá trị f nhỏ nhất để duyệt, nếu nó là đích, ta sẽ tiến hành việc xây dựng lại đường đi, còn không ta sẽ duyệt những điểm xung quanh nó, những điểm ‘hàng xóm’ này sẽ được thêm vào OpenList nếu nó thỏa mã tất cả các điều kiện. Trong các điều kiện mà một neighbor cần thỏa mãn ở trên bao gồm: Các tọa độ của nó không âm, không vượt quá giới hạn (bound) của map, không tính các node nằm theo phương chéo, hay không vi phạm lưu ý số 2 ở mục trên…

Nếu node ‘hàng xóm’ đó thỏa mãn các điều kiện đã lưu, ta sẽ khởi tạo node đó với giá trị tọa độ tương ứng, giá trị hàm g dựa theo giá trị hàm g của node đang được xét và hàm h được tính theo khoảng cách Mahattan.

Sau khi xét xong các node ‘hàng xóm’ node đang được xét sẽ bị loại bỏ ra khỏi OpenList, thêm vào ClosedList. Và thuật toán sẽ cứ thực hiện như vậy cho đến khi nó đạt được điều kiện dừng.

**Lưu ý**:

Để minh họa được trực quan, chúng em không sử dụng vòng lặp while như ở trên mà dùng lớp Timer và implement interface ActionListener để cập nhật thay đổi của các đối tượng. Sau mỗi khoảng thời gian là SPEED, thuật toán sẽ lại thực hiện hàm actionPerform(), cập nhật trạng thái của từng đối tượng và sau đó ta gọi đến hàm repaint() để nó gọi lại hàm paint(), vẽ lại các đối tượng đó lên màn hình và giúp ta dễ hình dung hơn. Về bản chất thì nó vẫn giống như vòng lặp while.

## *4.2. Nhận xét.*

*3.2.1. Về cách triển khai thuật toán:*

Đối với OpenList có n phần tử, ta chúng ta thực hiện các công việc

* Chèn một phần tử vào hàng đợi ưu tiên: O(n).
* Kiểm tra xem một node đã có trong OpenList hay chưa? Nếu chưa ta chỉ cần thêm node đó vào hàng đợi, nếu đã tồn tại ta xem giá trị hàm f(n) của node nào nhỏ hơn ta giữ lại: O(n).
* Lấy ra node có giá trị f(n) nhỏ nhất để tiếp tục mở rộng: O(1).
* Loại bỏ một phần tử ra khỏi OpenList khi các node xung quanh nó đã được duyệt: O(1).

Đối với ClosedList có n phần tử, ta cũng thực hiện:

* Kiểm tra xem node chờ duyệt đã có trong ClosedList hay chưa: O(n)
* Thêm node đã duyệt xong vào ClosedList: O(1).

Hơn nữa, trong mỗi lần lặp, ta cũng cần phải lặp qua các phần tử trong OpenList cũng như ClosedList để cập nhật trạng thái của nó và vẽ ra trên màn hình nhưng nhìn chung, chúng em nghĩ cách triển khai này là khá hợp lý cho thuật toán.

*3.2.2. Về thuật toán A\*:*

Hạn chế lớn nhất của thuật toán có lẽ là việc nó lưu trữ tất cả các node và đường đi qua các node mà nó đã xét để khi một node thêm vào được phát hiện là đích, nó sẽ lấy ra đường đi của node đó.

Trong trường hợp xấu nhất, khi mà điểm đích không thể được tìm thấy (không có đường đi đến đó hay nó ở rất rất xa, …) thuật toán sẽ phải duyệt tất cả các node trên bản đồ mà nó có thể đi tới được vậy độ phức tạp của thuật toán lúc này sẽ là O(n) với n là số đỉnh/cạnh trên đồ thị mà thuật toán có thể tới được.

Tuy vậy, nhưng nhìn chung nó vẫn là một thuật toán tìm kiếm nhanh chóng và hiệu quả. Do các node mà nó tìm kiếm đều là những node “có chọn lọc”. Bằng chứng là việc nó được ứng dụng rất nhiều vào trong lĩnh vực về khoa học máy tính.

# **V. Ứng dụng của giải thuật.**

Như đã đề cập ở trên, thuật toán A\* ban đầu được thiết kế để duyệt đồ thị và giúp robot có thể tự lập kế hoạch cho đường đi. Bên cạnh đó, nó còn được ứng dụng trong rất nhiều các nền tảng ứng dụng, trò chơi mà chúng ta vẫn thấy hàng ngày.

Điển hình nhất có lẽ là tựa game nối tiếng thế giới Liên Minh Huyền Thoại.

Graphical user interface, application

Description automatically generated

Về cơ bản, tựa game này yêu cầu các thao tác di chuyển rất nhiều, ngay khi người dùng click chuột vào một vị trí nào đó bất kỳ trên bản đồ, yêu cầu đặt ra là thuật toán phải tìm ra được đường đi đến đó ngay lập tức để nhân vật có thể đi theo hướng đó. Việc thuật toán xử lý chậm chễ, dù là trong khoảng thời gian rất nhỏ thôi khoảng 0.1 đến 0.2 giây thì nó cũng sẽ ảnh hưởng đến tâm lý người chơi cũng như là kết quả của cả ván đấu, nhất là khi bản đồ LMHT rất rộng và tựa game này còn thường xuyên được tổ chức ở các giải đấu lớn mang tầm cỡ khu vực cũng như quốc tế.

# **Kết Thúc.**

Qua phần trình bày chi tiết phía trên về thuật toán tìm đường đi ngắn nhất dựa trên giải thuật A\*, chúng ta cũng đã phần nào hiểu được và thấy được ứng dụng quan trọng của giải thuật A\* trong cuộc sống thực tế. A\* là thuật toán đã được Peter Hart, Nils Nilsson và Bertram Raphael tạo ra. Nó không chỉ cung cấp cho chúng ta sự đa dạng về thuật toán mà còn là một giải pháp tối ưu, giúp giảm thiểu thời gian trong các bài toán cần tìm đường đi ngắn nhất. Mặc dù ý tưởng triển khai nghe có vẻ rất đơn giản nhưng để bắt tay vào làm thì thuật toán A\* không phải chuyện đơn giản. Phía trên là phần trình bày của nhóm chúng em về thuật toán A\* áp dụng trong bài toán tìm đường đi ngắn nhất, trong quá trình viết báo cáo không thể tránh khỏi những sai sót cũng như cách diễn đạt chưa được rõ ý, chúng em rất mong nhận được sự thông cảm của thầy, cô và xin chân thành cảm ơn thầy cô!