**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**\*\*\***



**BÁO CÁO CUỐI KỲ**

**Bộ môn: Cấu trúc dữ liệu và giải thuật**

**Giảng viên:**

**PGS.TS Nguyễn Thị Hồng Minh**

**Thầy Đặng Trung Du**

**Thầy Trần Bá Tuấn**

**Đề tài: Tìm đường đi ngắn nhất sử dụng thuật toán A\***

***Sinh viên thực hiện:* Trần Duy Hiệp - 20001915**

**Kiều Quốc Ngọc - 20001956**

**Dương Tuấn Sơn - 20001970**

# **Hà Nội, tháng 11 năm 2023**

# **MỤC LỤC**

**LỜI MỞ ĐẦU**

**I. Giới thiệu bài toán**

**II. Giới thiệu về lý thuyết đồ thị**

1. Định nghĩa và phân loại đồ thị
2. Các khái niệm cơ bản trên đồ thị

**III. Giới thiệu thuật toán A\* (A-star)**

1. Thuật toán A\* là gì?

2. Heuristic là gì?

**IV. Mô tả và triển khai thuật toán A\*(A-star)**

1. Mã giả giải thuật A\*

2. Mô phỏng trên đồ thị

3. Triển khai thuật toán A\* trong lập trình Java

4. Đánh giá độ phức tạp thuật toán

5. Ứng dụng của cấu trúc dữ liệu vào thuật toán A\*

**V. Ưu, nhược điểm của thuật toán A\* (A-star)**

1. Ưu điểm

2. Nhược điểm

**VI. So sánh thuật toán A\* (A-star) và Dijkstra**

1. Thuật toán Dijkstra

2. Thuật toán A-star

3. Dijkstra có là trường hợp đặc biệt của A\* không?

**VII. Mô phỏng đồ họa**

# 

# 

# **LỜI MỞ ĐẦU**

Trong thực tế hiện nay, tối ưu hóa quãng đường di chuyển để tiết kiệm chi phí và thời gian là vấn đề luôn luôn được đặt lên hàng đầu trong nhiều lĩnh vực, đặt ra rất nhiều bài toán cần xử lý. Tuy nhiên các thuật toán tìm kiếm hiện tại như chúng ta được biết qua quá trình học tập học phần Cấu trúc dữ liệu và Thuật toán là Dijkstra chỉ xây dựng các tuyến đường có thể dẫn đến đích bằng cách xét tất cả những nút trong đồ thị, chưa tối ưu được quãng đường di chuyển. Chính vì thế, qua tìm hiểu và phân tích kỹ lưỡng, nhóm 9 lựa chọn thuật toán A\* (A sao) để giải quyết một phần vấn đề đó cũng như làm đề tài chính cho dự án cuối kỳ của nhóm.

Nhóm 9 xin gửi lời cảm ơn sâu sắc đến PGS.TS Nguyễn Thị Hồng Minh cùng Thầy Đặng Trung Du và Thầy Trần Bá Tuấn - Giảng viên Bộ môn Tin học và phụ trách giảng dạy học phần Cấu trúc dữ liệu và Giải thuật đã hướng dẫn và giúp đỡ chúng em rất nhiều trong quá trình tiếp cận, tìm hiểu và thực hiện những yêu cầu trong lộ trình môn học. Qua những bài giảng, những giờ chữa bài về nhà bổ ích của thầy cô, chúng em được tăng thêm vốn hiểu biết về thuật toán, cách tổ chức bài làm và khắc phục những lỗi cố hữu trong trình bày và tư duy lập trình. Dưới đây, nhóm 9 xin trình bày nội dung bản báo cáo đề tài: “Tìm đường đi ngắn nhất sử dụng thuật toán A\* “.

## 

## 

**I. Giới thiệu bài toán**

Hiện nay, tại các thành phố lớn hay những khu đô thị - những nơi có dân cư sinh sống với mật độ lớn, cơ sở hạ tầng và giao thông ngày càng phát triển kéo theo nhiều vấn đề cần giải quyết như: tắc đường, cơ sở hạ tầng xuống cấp,… Trong những năm vừa qua, đối với người dân trên thế giới nói chung và Việt Nam nói riêng, những giải pháp để khắc phục vấn đề tắc đường luôn được ưu tiên đặt lên hàng đầu. Vấn đề ấy không còn quá lạ đối với mỗi chúng ta nhưng thời gian ngắn trở lại đây, tình hình thế giới biến động khiến cho giá xăng, dầu đột ngột tăng mạnh. Giá xăng ngày càng cao là nỗi lo lắng cho mỗi người dân về phương tiện di chuyển. Chính vì điều đó, việc tìm con đường đi ngắn nhất, tiết kiệm thời gian và chi phí đi lại giờ đây lại chính là ưu tiên số một và đặt ra cho chúng ta một bài toán cần giải quyết. Nhóm 9 sẽ dùng thuật toán A\* để giải quyết một phần của vấn đề qua trình bày lý thuyết và minh họa bằng đồ họa.

**II. Giới thiệu về lý thuyết đồ thị.**

1. **Định nghĩa và phân loại đồ thị**

Một đồ thị là một cấu trúc rời rạc gồm tập hợp các đỉnh và các cạnh nối giữa các đỉnh đó. Có thể mô tả đồ thị theo cách hình thức như sau:

*G* = (*V*, *E*)

tức là đồ thị *G* có tập các đỉnh (vertices) là *V*, tập các cạnh (Edges) là  *E*. Có thể hiểu *E* là tập hợp các cặp (*u*,*v*) với *u* và *v* là hai đỉnh thuộc *V*.

Có thể phân loại đồ thị *G* theo tính chất của tập cạnh như sau:

* *G* được gọi là ***đơn đồ thị*** nếu như giữa hai đỉnh (*u*,*v*) của *V* có nhiều nhất một cạnh trong *E* nối từ *u* tới *v*.
* *G* được gọi là ***đa đồ thị*** nếu như giữa hai đỉnh (*u*,*v*) của *V* có thể có nhiều hơn 11 cạnh nối trong *E* nối từ *u* tới *v*. Hiển nhiên đơn đồ thị cũng là đa đồ thị.
* *G* được gọi là ***đồ thị vô hướng*** (undirected graph) nếu như các cạnh trong *E* là không định hướng, tức là cạnh (*u*,*v*) là ***cạnh hai chiều***.
* *G* được gọi là ***đồ thị có hướng*** (directed graph) nếu như các cạnh trong *E* là có định hướng, tức là có thể tồn tại cạnh nối từ u tới v nhưng chưa chắc đã tồn tại cạnh nối từ v tới u. Trên đồ thị có hướng, các cạnh sẽ được gọi là các ***cung***. Đồ thị vô hướng cũng có thể coi là đồ thị có hướng, nếu như ta

coi cạnh ( u,v) bất kỳ tương ứng với hai cung (u →v) và ( v → u).

* G là đồ thị có trọng số : nếu các cạnh /cung của nó có giá trị

1. **Các khái niệm cơ bản trên đồ thị .**

**a, Các liên thuộc , đỉnh kề, bậc** .

Đối với đồ thị vô hướng G = (V, E) , xét cạnh e = (u, v) € E. Ta nói rằng hai đỉnh u và v kề nhau ( adjacent), và cạnh e này thuộc (incident) với hai đỉnh u và v.

Với 1 một đỉnh u thuộc đồ thị , đỉnh bậc (degree) , ký hiệu deg (u) là số cạnh liên thuộc với u . Trên đồ thị số cạnh liên thuộc với u cũng chính là số đỉnh kề với u.

**b, Đường đi và chu trình.**

Một đường đi P có độ dài là k từ đỉnh v0 tới đỉnh vk tới tập đỉnh {v0, v1, v2,…, vk} sao cho ( vi-1 , vi) € E , ∀*i*:1≤*i*≤*k* . Khi đó ta nói đường đi này bao gồm các đỉnh {v0, v1, v2,…, vk} và các cạnh {(v0, v1), (v1, v2).…(vk-1, vk)} ; và v0 đến được vk  thông qua đường đi P.

Đường đi được gọi là đường đi đơn giản (simple path) nếu tất cả các đỉnh trên đường đi đó đều phân biệt . Đường đi được gọi là đường đi đơn nếu như không có cạnh trên đường đi đó đi qua hơn 1 lần .

Một *đường đi con (subpath)* ′*P*′ của *P* là một đoạn liên tục các đỉnh và cạnh dọc theo đường đi *P*.

Đường đi P gọi là chu trình (circuit) nếu như v0 = vk . Chu trình P gọi là chu trình đơn giản *(simple circuit)* nếu như {v0, v1, v2,…, vk} đôi một khác nhau . Chu trình mà trong đó không có cạnh nào đi qua hơn một lần gọi là chu trình đơn.

**III. Giới thiệu thuật toán A\* (A-star)**

Năm 1964, Nils Nilsson phát minh ra một phương pháp tiếp cận dựa trên khám phá để tăng tốc độ của thuật toán Dijkstra. Thuật toán này được gọi là A1. Năm 1967, Bertram Raphael đã cải thiện đáng kể thuật toán này, nhưng không thể hiển thị tối ưu và đặt tên thuật toán này là A2.

Năm 1968, Peter E. Hart đã giới thiệu một đối số chứng minh thuật toán A2 của ông khi sử dụng thuật toán này với một đánh giá heuristic thích hợp sẽ thu được kết quả tối ưu. Chứng minh của ông về thuật toán cũng bao gồm một phần cho thấy rằng các thuật toán A2 mới là thuật toán tốt nhất có thể được đưa ra các điều kiện. Do đó, ông đặt tên cho thuật toán mới là A\* (A sao, A-star).

**1. Thuật toán A\* là gì?**

Trong khoa học máy tính, A\* (được đọc là A sao) là thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm đường đi từ một nút khởi đầu tới một nút đích cho trước. Thuật toán này sử dụng một đánh giá heuristic để đánh giá từng nút theo ước lượng về tuyến đường đi tốt nhất đi qua nút được xét. Thuật toán A\* duyệt các nút theo thứ tự đánh giá của heuristic.

**2. Heuristic là gì?**

Heuristic là phương pháp giải quyết vấn đề dựa trên phỏng đoán, ước chừng, kinh nghiệm, trực giác để tìm ra giải pháp gần như là tốt nhất và nhanh chóng.

Heuristic được tính toán bằng hàm Heuristic - hàm ứng với mỗi trạng thái hay mỗi sự lựa chọn một giá trị ý nghĩa của vấn đề. Dựa vào giá trị hàm này, ta thực hiện lựa chọn hành động.

Tổng chi phí của một nút (node) được thể hiện qua hàm f(x) được tính như sau:

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

***Trong đó:***

* g(x) là hàm chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua.
* h(x) là hàm đánh giá Heuristic về chi phí nhỏ nhất để đi đến đích từ đỉnh x (hiện tại).
* f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.

**IV.** **Mô tả và triển khai thuật toán A\*(A-star)**

**1. Mã giả giải thuật A\***

Tạo một danh sách C là tập hợp nodes đã được xét đến

Tạo một danh sách O là tập hợp nodes chưa được xét đến và giá trị f tương ứng

**while** Tập O không rỗng :

Chọn 1 node n từ O với giá trị f tốt nhất

**if** n là target node :

**return** Tìm được đường đi

**for** Toàn bộ m là neighbor của n:

**if** (m không thuộc tập C) **and** (m không thuộc tập O):

Thêm m vào O, thiết lập n là cha của m

Tính g(m) và f(m) đồng thời lưu lại

**else :**

**if** f(m) từ lần lặp cuối cùng tốt hơn g(m) ở lần lặp hiện tại :

Thiết lập n là cha của m

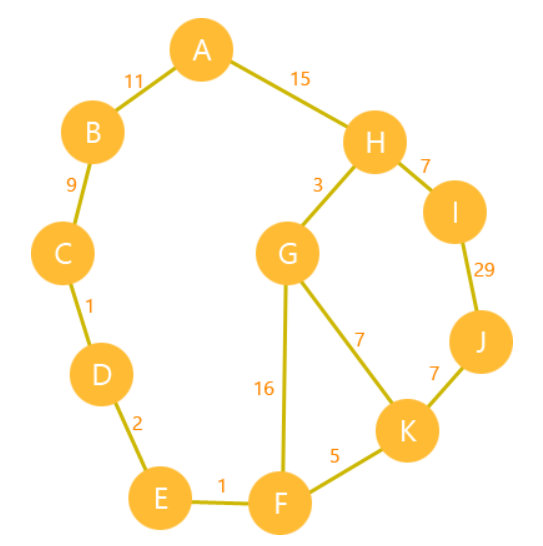
Cập nhật g(m) và f(m)

**if** m thuộc tập C :

Chuyển m vào tập O

Chuyển n từ tập O đến tập C

**return** Không tìm được đường đi

**2. Mô phỏng thuật toán trên đồ thị**

**Bài toán:**Tìm đường đi ngắn nhất từ A đến K. ***Biết:*** h(A) = 60 / h(B) = 53 / h(C) = 36 / h(D) = 35 / h(E) = 35 / h(F) = 19 / h(G) = 16 / h(H) = 38 / h(I) = 23 / h(J) = 0 / h(K) = 7

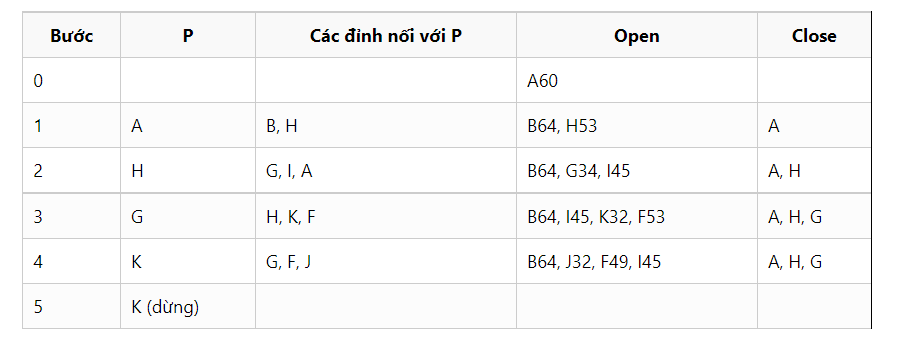
*(h(x) là hàm đánh giá Heuristic về chi phí nhỏ nhất để đi đến đích từ đỉnh x (hiện tại)).*

**Giải thuật:**

- Đỉnh bắt đầu: **A**

- Đỉnh kết thúc: **K**

Ước lượng chi phí từ đỉnh đầu cho đến đỉnh kết thúc khi qua đỉnh hiện tại, trong đó g là chi phí từ đỉnh đầu đến đỉnh hiện tại.

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

**Bước 0:** Đỉnh bắt đầu là đỉnh A, thêm A vào Open

Open = {**A**}

Close = {}

**Bước 1:** Đỉnh đang xét: **A**

f(A) = g(A) + h(A) = 0 + 60 =60

Đỉnh lân cận của đỉnh đang xét: B và H => Thêm B và H vào Open.

Open = {A, B, H}

f(B) = g(B) + h(B)= 11+53 = 64

f(H) = g(H)+ h(H) = 15+38 = 53

Loại A khỏi Open và thêm A vào Close.

Close = {A}

Có f(H) < f(B) => Đỉnh xét tiếp theo p = H

**Bước 2:** Đỉnh đang xét: **H**

f(H) = 53

Đỉnh lân cận của đỉnh đang xét: G, I và A => Thêm G và I vào Open.

Open = {B, H, G, I}

f(G) = g(G) + h(G)= 18 + 16= 34

f(I) = g(I)+ h(I) = 22 + 23 = 45

Bỏ H khỏi Open và thêm H vào Close.

Close = {A, H}

Có f(G) < f(I) < f(B) => Đỉnh xét tiếp theo p = G

**Bước 3:** Đỉnh đang xét: **G**

f(G) = 34

Đỉnh lân cận của đỉnh đang xét: H, K, F => Thêm K, F vào Open.

Open = {B, G, I, K, F}

f(K) = g(K) + h(K)= 25 + 7 = 32

f(F) = g(F)+ h(F) =34 + 19 = 53

Loại G khỏi Open và thêm G vào Close.

Close = {A, H, G}

Có f(K) < f(I) < f(F) < f(B) => Đỉnh xét tiếp theo p = K

**Bước 4:** Đỉnh đang xét: **K**

=> Đến K (target) => **Dừng thuật toán.**

**3. Triển khai thuật toán A\* trong lập trình Java**

Method **aStar()** trong class **Graph** thực hiện thuật toán A\*

Khởi tạo đồ thị, sau đó khởi tạo openList (**open**) và closeList (**close**). Đồng thời, thiết lập (set) giá trị mặc định của g và h trong các nút (node).

Tính toán giá trị heuristic h (**calculateHeuristic()**) của nút đầu (**start**) và thêm nút đầu vào **open**. Sau đó thực hiện vòng lặp **while()** khi **open** khác rỗng.

**Vòng lặp:**

**Bước 1:** Chọn nút có giá trị f thấp nhất và cho nút đó làm currentNode, nếu currentNode trùng với điểm đích thì dừng thuật toán.

**Bước 2:** Duyệt qua các neighbor của currentNode,

* Nếu như neighbor không có trong open và close thì thực hiện tính toán neighbor.g và neighbor.f. Sau đó, ta thiết lập neighbor vừa xét thành con của currentNode và thêm vào open.

***Thể hiện code:***

if (!open.contains(neighbor) && !close.contains(neighbor)) {

neighbor.g = newG;

neighbor.f = newG + neighbor.calculateHeuristic(targetNode);

neighbor.setParent(currentNode);

open.add(neighbor);

}

* Nếu như neighbor đã có trong open và newG < neighbor.g thì ta thực hiện gán neighbor.g = newG và tính toán lại neighbor.f.
* Nếu như neighbor đã có trong close thì thực hiện loại neighbor ra khỏi close và thêm vào open để xét lại.

=> Sau đó, ta thiết lập neighbor vừa xét thành con của currentNode.

***Thể hiện code:***

if (newG < neighbor.g) {

neighbor.g = newG;

neighbor.f = newG + neighbor.calculateHeuristic(targetNode);

if (close.contains(neighbor)) {

open.add(neighbor);

close.remove(neighbor);

}

neighbor.setParent(currentNode);

}

**Bước 3:** Thực hiện loại currentNode ra khỏi open và thêm vào close.

open.remove(currentNode);

close.add(currentNode);

**4. Đánh giá độ phức tạp của thuật toán**

Bỏ qua các trường hợp đặc biệt, độ phức tạp của A\* có thể được xấp xỉ dựa trên số lượng neighbor của mọi nút trên đường dẫn ngắn nhất. Giả sử rằng mọi nút đều có hầu hết các neighbor b và con đường ngắn nhất đi qua d node. Độ phức tạp của A\*: O(bd)

Trong trường hợp xấu nhất, nó sẽ thực hiện duyệt toàn bộ các nút, khi đó độ phức tạp của thuật toán sẽ là : O(N2)

**5, Ứng dụng của cấu trúc dữ li****ệu vào thuật toán A\*.**

Hàng đợi ưu tiên (PriorityQueue): được sử dụng để lưu trữ các đỉnh chưa xét (open set) và các đỉnh đã xét (close set), giúp đảm bảo rằng các đỉnh được duyệt qua sẽ được xét theo thứ tự ưu tiên dựa trên một giá trị độ ưu tiên (đôi khi được gọi là "chi phí ước lượng" hoặc "hạn chế").

Đồ thị (Graph) : Cấu trúc đồ thị được sử dụng để biểu diễn mô hình lưới với các nút và các cạnh giữa chúng. Điều này hỗ trợ thuật toán A\* trong việc tìm kiếm đường đi từ nút bắt đầu đến nút đích .

LinkedList và List : dùng để lưu trữ và quản lý danh sách các nút trong đồ thị.

Swing : Cấu trúc Swing dùng để xây dựng giao diện người dùng (GUI) của chương trình.

**V. Ưu, nhược điểm của thuật toán A\* (A-star)**

**1. Ưu điểm**

A\* là một thuật giải có thể biến đổi khéo léo tùy vào tình thế, tổng quát, trong đó hàm chứa cả tìm kiếm theo chiều sâu (DFS), tìm kiếm theo chiều rộng (BFS) và những nguyên lý Heuristic khác.

A\* nhanh chóng tìm được giải pháp tối ưu với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói A\* chính là một trong những thuật giải tối ưu ứng dụng phương pháp Heuristic.

**2. Nhược điểm**

Thuật toán A\* thường gây tốn bộ nhớ vì nó cần lưu lại những trạng thái mỗi bước khi mà nó đi qua, điều này chúng ta có thể thấy tương tự như với thuật toán tìm kiếm theo chiều rộng (BFS).

**VI. So sánh thuật toán A-star và Dijkstra**

**1. Thuật toán Dijkstra**

* Có một hàm để tính chi phí, hàm này thể hiện giá trị chi phí thực từ điểm bắt đầu đến từng node:

**f(x) = g(x)**

* Thuật toán Dijkstra tìm con đường ngắn nhất từ ​​điểm bắt đầu đến mọi node khác cần xét chỉ bằng cách xem xét chi phí thực khi thực hiện.

+ Thuật toán Dijkstra: xây dựng tuyến đường dẫn đến đích bằng cách xét tất cả các nút trong đồ thị dẫn đến chưa tối ưu được quãng đường di chuyển.

→ Sử dụng thuật toán A\* để giải quyết vấn đề mà Dijkstra đang gặp phải.

**2. Thuật toán A-star**

Thuật toán Astar dùng hai hàm g(x) và h(x) để thực hiện tính chi phí:

* **g(x)**: hàm g(x) trong thuật toán A\* có chức năng giống như trong thuật toán Dijkstra. Hàm này là hàm tính chi phí thực để đạt được một node x.
* **h(x)**: hàm h(x) trong thuật toán A\* là một hàm heuristic, hàm này tính chi phí ước lượng từ node x đến đỉnh kết thúc (target). Giá trị heuristic của hàm h(x) không bao giờ được tính cao hơn chi phí. Nghĩa là, chi phí thực để đến được đỉnh kết thúc (target) từ một nút x nào đó phải lớn hơn hoặc bằng giá trị hàm h(x).
* Tổng chi phí của mỗi node được tính bằng hàm **f(x)**:

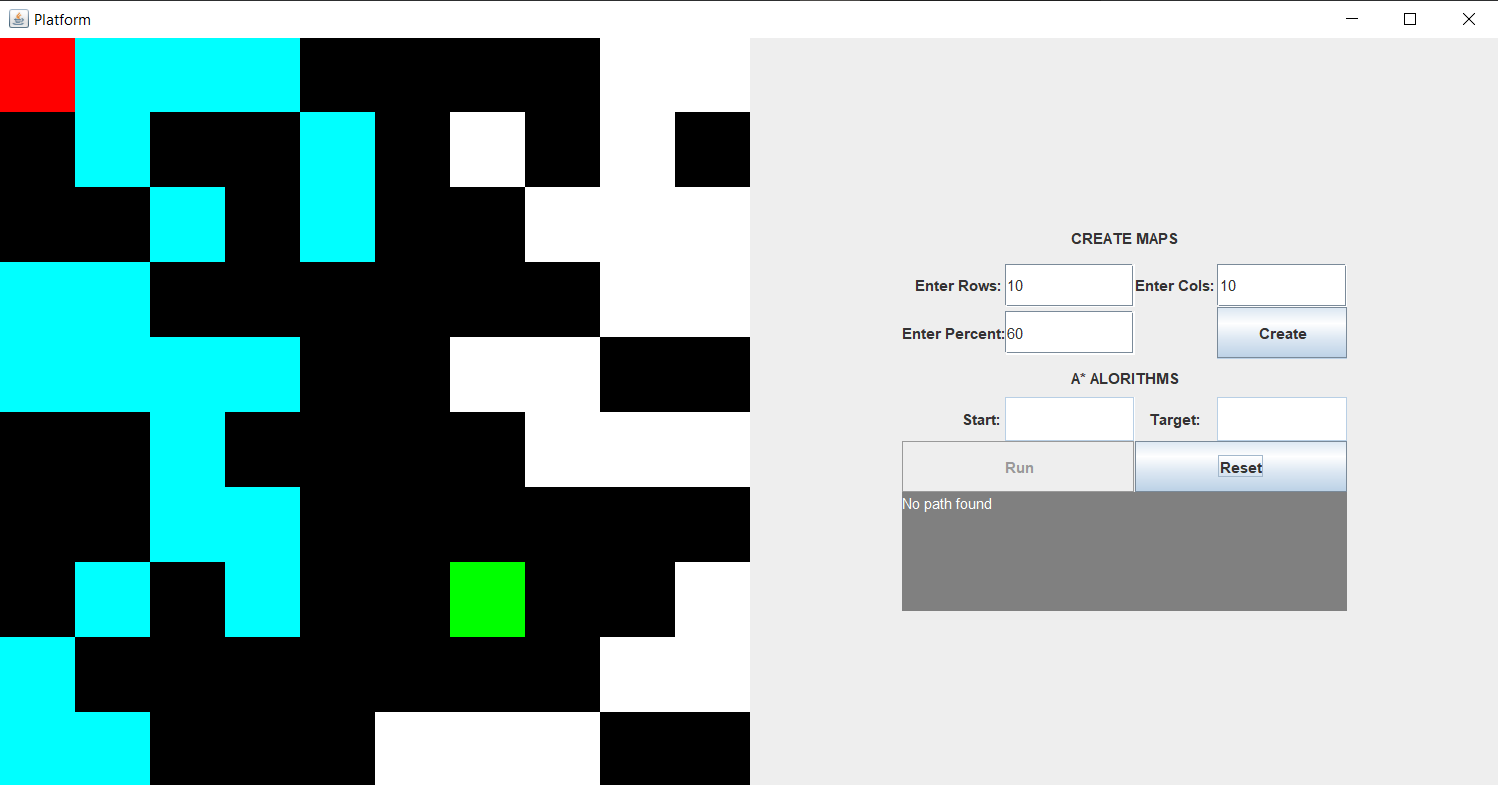
**f(x) = g(x) + h(x)**

* Thuật toán A\* chỉ mở rộng một node nếu nó có khả năng tốt. Nó sẽ tập trung để tiếp cận điểm đích (target) từ nút (node) hiện tại, không tiếp cận các nút khác. Nếu như heuristic tốt, thì đó là phương pháp tối ưu để thực hiện giải bài toán.

**3. Dijkstra có là trường hợp đặc biệt của A\* không?**

* Có thể nói Dijkstra là một trường hợp đặc biệt của A\* vì như đã được định nghĩa ở trên Dijkstra có thể hiểu là thuật toán A\* khi chi phí từ tất cả các đỉnh (hàm heuristic) đến đỉnh kết thúc đều bằng 0.
* Trong trường hợp thuật toán A\* không thể tìm được đường đi tới điểm đích, nó sẽ thực hiện xét tất cả những nút trong đồ thị. Khi đó, thuật toán A\* thực hiện như thuật toán Dijkstra.

**VII. Mô phỏng đồ họa**

**Không tìm được đường đi**

**Tìm được đường đi**

A screenshot of a computer screen

Description automatically generated**Không nhập đủ startID và targetID**