**TÌM KIẾM ĐƯỜNG DẪN ĐỂ HAI NGƯỜI GẶP NHAU NHANH NHẤT TRÊN ĐỒ THỊ**

**Võ Ngọc Nghĩa1, Nguyễn Phan Nhật Tú1, Huỳnh Nhựt Thiên1**

*1Trường đại học Sư phạm Kỹ thuật TP.HCM*

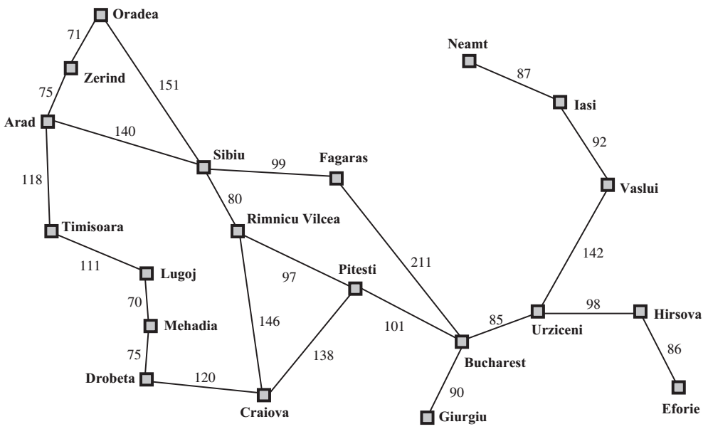
**TÓM TẮT**

Hiện nay nhu cầu đường đi, hướng dẫn lộ trình đang là nhu cầu cấp thiết của hầu như tất cả các đối tượng. Việc dẫn đường được sử dụng trong nhiều lĩnh vực khác nhau như du lịch, giao thông, thương mại,… Để giải quyết vấn đề tìm kiếm đường dẫn gặp nhau nhanh nhất của hai đối tượng bất kì đựa trên các phương pháp tìm kiếm cùng với sự phát triển của trí tuệ nhân tạo sẽ giúp đường đi đạt hiệu quả tốt nhất.Vì vậy, vấn đề được đặt ra là với một đồ thị hoặc bảng đồ cho trước hãy tìm đường đi nhanh nhất với chi phí và thời gian thấp giữ hai vi trí bất kì trên đó.Từ kết quả đó con người có thể chế tạo ra các ứng dụng hữu ích dùng trong sinh hoạt cuộc sống thực tế cụ thể hơn là các ứng dụng bản đồ mạng tính chất chỉ đường. Đồ án áp dụng các kiến thức đã học, một số ràng buộc được tìm ra trong quá trình phát triển cùng với sử dụng tư duy của môn trí tuệ nhân tạo để giải thuật toán.

Từ khóa: dẫn đường, trí tuệ nhân tạo trong việc tìm kiếm đường đi, ứng dụng bản đồ, thuật toán tìm kiếm

1. **GIỚI THIỆU**

Đồ án đặt ra một trường hợp cụ thể để giúp chúng ra xác định được đối tượng, tính chất và tìm ra phương pháp nguyên cứu. Cho đồ thị như hình:



***Hình 1.*** *Đồ thị của đề tài*

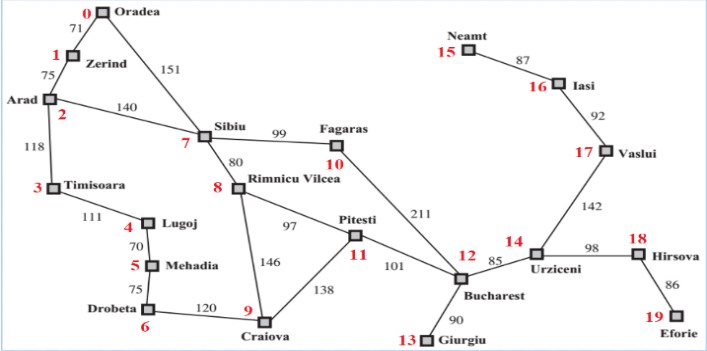
Giả sử có 2 người, mỗi người ở 2 vị trí bất kỳ trên đồ thị. Mỗi bước di chuyển 2 người đồng thời đi đến một đỉnh bất kỳ liền kề với đỉnh hiện tại. Thời gian di chuyển giữa 2 đỉnh i,j bằng khoảng cách d(i,j) của 2 đỉnh. Trong 1 bước di chuyển, 2 người cần phải đợi nhau để tiếp tục đi tiếp: người 1 đi đến 1 đỉnh thì phải đợi người 2 đi đến 1 đỉnh thì mới tiếp tục cho 2 người đi (có thể gọi điện để xác nhận). Hãy tìm kiếm đường dẫn để 2 người gặp nhau nhanh nhất trên đồ thị. Qua bài toán cụ thể này, việc giải thực tế và tìm ra được từng chi tiết ràng buộc có thể gặp trong quá trình nghiên cứu, xây dựng lời giải tối ưu sẽ trợ nên dễ dàng hơn. Sinh viên có thể so sánh với các bài toán tìm đường khác hoặc những bài toán liên quan đến đồ thị đã được học, đây cũng được xem là áp dụng bài học vào thực tế và đời sống.

Các phần còn lại của bài báo cáo được tổ chức như sau. Phần 2 trình bày về phương pháp tìm hiểu và thực hiện, bao gồm các thuật toán sử dụng, mã giả và các ràng buộc cho các trường hợp đặc biệt.Phần 3 trình bày kết quả của quá trình thực hiện và một số ví dụ kết quả mà chương trình chạy được.Phần 4 tổng hợp và kết luận về đồ án này.

1. **PHƯƠNG PHÁP**

**2.1. Cách nghiên cứu và thực hiện**

Từ đồ thị đã cho, đối tượng sẽ được giả sử là hai người đứng tại hai đỉnh bất kì trên đồ thị. Từ đó tìm đường nhanh nhất và thời gian chờ thấp nhất để hai người gặp được nhau.Bước đầu tiên, nhóm sẽ xem xét đồ thị của đề tài đề thiết lập ma trận kế đựa theo chi phí và các đỉnh kề nhau theo thứ tự do nhóm tự thiết lập. Nhóm sử dụng thuật toán Bidirectional search để bước đầu xác định được đường đi của hai người di chuyển trên đồ thị sao cho cả hai gặp nhau. Sau đó sẽ nguyên cứu về chi phí và thời gian đợi khi hai người cùng đi theo yêu cầu đề tài.Thuật toán Best First search dùng để tối ưu hóa đường đi, sao nó cùng số bước cùng với chi phí và thời gian chờ đạt hiệu quả tốt nhất.Tìm ra các trường hợp đặc biệt như khi hai người bắt đầu đi từ cùng một điểm (VD: 0 – 0, 1 – 1, 2 – 2,…); hai đỉnh kề nhau;… Để xác định được các ràng buộc của thuật toán bằng cách thử tất cả các trường hợp mà từ một đỉnh đi đến 19 đỉnh còn lại.Để có các sự lựa chọn hướng đi, nhóm đã tiến hành tạo mảng chứa các trường hợp đỉnh đó có thể đi và xét hướng đi của đỉnh theo mảng đó. Các dữ kiện sẽ là chi phí thấp, thời gian chờ ít và số bước đi nhỏ.



***Hình 2.*** *Thứ tự các đỉnh của đồ thị*

**2.2. Mã giả của các thuật toán và phương pháp được sử dụng**

**Funtion BidirectionalSearch(điểm xuất phát người 1, điểm xuất phát người 2, khoảng cách các điểm với nhau)**

If xuất phát người 1 = điểm xuất phát người 2

kq = true

bool visited\_s[20], visited\_t[20];

int parent\_s[20], parent\_t[20];

// danh sách các thành phố có thể đi từ 1 điểm

list<int>\* s\_queue;

list<int>\* t\_queue;

//Khởi tạo trạng thái ban đầu, những điểm có thể đi từ vị trí đầu tiên của 2 người

Khoitao(s\_queue, t\_queue, parent\_s, parent\_t, fx, visited\_s, visited\_t)

While (queue\_s và queue\_t khác rỗng)

bestFirstSearch(s\_queue,t\_queue,parent\_s,parent\_t,fx,visited\_s,visited\_t );

//Kiểm tra 2 người có gặp nhau chưa

intersectNode = isIntersecting(visited\_s, visited\_t, s\_queue, t\_queue, parent\_s, parent\_t);

if (dungct == true)

return -1;

if intersectNode != -1

xuất ra đường đi, thời gian chờ và tổng thời gian 2 người gặp nhau

exit(0)

return -1

Nếu chạy xong chương trình nhưng kết quả là -1 thì chạy lại nhưng đổi hàm isIntersecting thành isIntersecting1

Nếu vẫn không có kết quả thì xuất ra không tìm được đường

**Khoitao(s\_queue, t\_queue, parent\_s, parent\_t, fx, visited\_s, visited\_t)**

Foreach i in đường đi tiếp theo của current\_s

If chưa từng đi qua

Đưa i vào queue\_s[current\_s]

Foreach i in đường đi tiếp theo của current\_t

If chưa từng đi qua

Đưa i vào queue\_t[current\_t]

**bestFirstSearch(s\_queue,t\_queue,parent\_s,parent\_t,fx,visited\_s,visited\_t );**

if kq=true

return;

//Lấy 2 điểm sao cho thời gian chờ thấp nhất để mở rộng tiếp

get(queue\_s, queue\_t, parent\_s, parent\_t, fx, visited\_s, visited\_t)

if kq = true

return;

if dungct = true

return;

Foreach i in đường đi tiếp theo của current\_s

If chưa từng đi qua

Đưa i vào queue\_s[current\_s]

Foreach i in đường đi tiếp theo của current\_t

If chưa từng đi qua

Đưa i vào queue\_t[current\_t]

**get(queue\_s, queue\_t, parent\_s, parent\_t, fx, visited\_s, visited\_t)**

if queue\_s và queue\_t khác rỗng

dungct=true;

return;

if queue\_s[current\_s] và queue\_t[current\_t] khác rỗng

foreach i in queue\_s[current\_s]

foreach j in queue\_t[current\_t]

if i = j

kq=true

đánh dấu đã đi qua i và j, tính thời gian chờ của cả 2

return

tính và lấy điểm vitri\_i và vitri\_t có thời gian chờ min nhỏ nhất

Xóa vitri\_s và vitri\_t ra khỏi queue[current\_s] và queue[current\_t]

level\_s++ và level\_t++

` gán parent và tính thời gian đợi

Gán current\_s và current\_t = vitri\_s và vitri\_t

Else if queue\_s[current\_s] và queue\_t[current\_t] đều bằng rỗng

Gán current = parrent[current] cho tới khi queue[current] khác rỗng, cho cả s và t

Nếu level của s và t chênh lệch nếu s lớn hơn thì current\_s=parent\_s[current\_s] cho tới khi level cả 2 bằng nhau nhưng thêm vào queue của s giá trị đã xóa vì để s có thể đi lại đường cũ. Nếu level t > s thì ngược lại

Quay lại trường hợp cả queue\_s[current\_s] và queue\_t[current\_t] khác rỗng

Else if queue\_s[current\_s] bằng rỗng

Gán current\_s = parent\_s[current\_s] cho tới khi queue\_s[current\_s] khác rỗng

Gán current\_t = parent\_t[current\_t] cho tới khi level của s và t bằng nhau nhưng thêm vào queue\_t giá trị đã xóa trước đó để t cso thể quay lại đường cũ

Quay lại trường hợp cả queue\_s[current\_s] và queue\_t[current\_t] khác rỗng

Else if queue\_t[current\_t] bằng rỗng

Giống trường hợp queue\_s[current\_s] bằng rỗng nhưng ngược lại

//Hàm kiểm tra s và t có gặp nhau chưa

**isIntersecting(visited\_s, visited\_t, s\_queue, t\_queue, parent\_s,parent\_t)**

for i chạy hết tất cả các điểm ( thành phố) để kiểm tra

If cả s và t đã đi qua và level của s và t tại điểm (thành phố đó) bằng nhau

Return i

Else if s và t đã đi qua nhưng level khác nhau

If level tại điểm đó của s khác level hiện tại của s

Cả s và t lùi lại 1 bước (current=parent[current]) nhưng t vẫn thêm lại đưởng đi đó vào queue\_t vì t có quyền đi lại đường đó

Else if level tại điểm đó của t khác level hiện tại của t

Cả s và t lùi lại 1 bước (current=parent[current]) nhưng s vẫn thêm lại đưởng đi đó vào queue\_s vì s có quyền đi lại đường đó

Return -1;

**isIntersecting1(visited\_s, visited\_t, s\_queue, t\_queue, parent\_s,parent\_t)**

for i chạy hết tất cả các điểm ( thành phố) để kiểm tra

If cả s và t đã đi qua và level của s và t tại điểm (thành phố đó) bằng nhau

Return i

Return -1;

1. **KẾT QUẢ**

Tìm kiếm đường dẫn để hai người gặp nhau nhanh nhất trên đồ thị với thời gian chờ thấp nhất.Kết quả sẽ hiển thị các thông tin như sau: Hai đỉnh bất kì trên đồ thị, đỉnh gặp nhau, đường đi của hai người để có thể gặp nhau, thời gian đợi của người thứ nhất, thời gian đợi của người thứ hai và cuối cùng là tổng thời gian để người thứ nhất và người thứ hai gặp nhau.

***Bảng 3.*** *Ví dụ một số kết quả*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hai đỉnh | Đỉnh gặp nhau | Đường đi | Thời gian đợi của người thứ nhất | Thời gian đợi của người thứ hai | Tổng thời gian |
| 0 và 10 | 7 | 0 🡪 7 🡨 10 | 0 | 52 | 151 |
| 0 và 15 | 11 | 0 🡪 1 🡪 2 🡪 7 🡪 8 🡪 11 🡨 12 🡨 14 🡨 17 🡨 16 🡨 15 | 44 | 0 | 507 |
| 3 và 19 | 8 | 3 🡪 4 🡪 5 🡪 6 🡪 9 🡪 8 🡨 11 🡨 12 🡨 14 🡨 18 🡨 19 | 38 | 93 | 560 |
| 2 và 13 | 8 | 2 🡪 1 🡪 0 🡪 7 🡪 8 🡨 9 🡨 11 🡨 12 🡨 13 | 111 | 13 | 488 |
| 1 và 2 | 9 | 1 🡪 2 🡪 3 🡪 4 🡪 5 🡪 6 🡪 9 🡨 11 🡨 8 🡨 7 🡨 0 🡨 1 🡨 2 | 90 | 47 | 659 |

1. **KẾT LUẬN**

Trong đồ án đã áp dụng các kiến thức đã học như Bidirectional search và Best first search và tư duy của môn trí tuệ nhân tạo để giải thuật toán. Chúng ta có thể biết được những cách mà hai người ở những nơi khác nhau có thể gặp nhau. Nó có thể phát triển và hoàn thiện như các ứng dụng bản đồ chỉ đường(**HERE Maps, Google Maps, VOV bản đồ giao thông,..)**, ứng dụng gọi xe(Be, Grab, Go-Viet, VATO,…), ứng dụng xe buýt(bus map) và một số ứng dụng đặt vé di chuyển khác rất phổ biến và tiện ích. Ôn lại các kiến thức đã được học ở môn học, thấy được ứng dụng thức tế của nó đối với đời sống và công việc từ đó định hướng những gì cần làm và phải làm khi là một lập trình viên.

**TÀI LIỆU THAM KHẢO**

[1] <https://www.geeksforgeeks.org/>, Bidirectional Search, <https://www.geeksforgeeks.org/bidirectional-search/>

[2] <https://en.wikipedia.org/wiki/Main_Page>, Bidirectional Search, <https://en.wikipedia.org/wiki/Bidirectional_search>

[3] Tất cả các code và bài giảng của môn Trí Tuệ Nhân tạo do GV.Huỳnh Xuân Phụng giảng dạy