**“TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA HAI THÀNH PHỐ”**

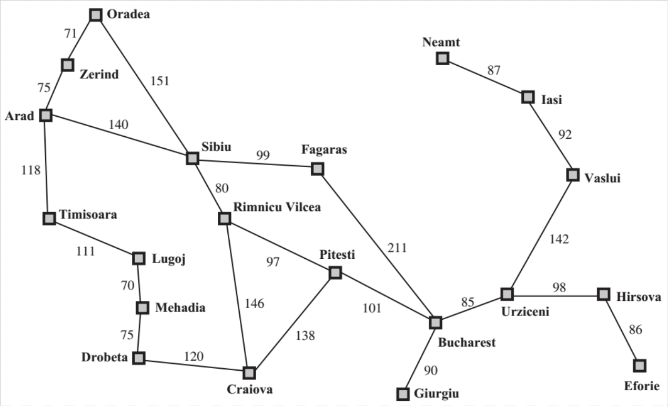
**Đặng Quốc Trung, Ngô Anh Quốc, Lê Văn Hoàn**

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

**TÓM TẮT**

*Ngày nay, với sự phát triển mạnh mẽ và tiên tiến của nước ta, các tuyến đường giao thông ngày càng được cải thiện và mở rộng nhằm phục vụ cho nhu cầu đi lại của con người. Nhưng chính sự cải thiện đó làm cho việc chọn ra các tuyến đường đi tốt nhất lại trở nên rất khó khăn. 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), Chính vì thế dù có nhiều tuyến đường để 2 người đó gặp nhau đi chăng nữa thì họ cũng không thể biết được đâu là sẽ tuyến đường* ***ngắn nhất*** *và tốn ít chi phí nhất ? Để có thể giải quyết được vấn đề đó thì nhóm em đã chọn đề tài này để xây dựng nên chương trình “Tìm kiếm đường đi* ***ngắn nhất*** *giữa hai thành phố” để giúp cho con người có thể chọn được đường đi tối ưu nhất về thời gian và cả chi phí, bằng cách sử dụng 2 thuật toán tìm kiếm là* ***Greedy best first search*** *và* ***A\**** *và điều đặc biệt ở đây là thuật toán A\* là thuật toán tối ưu nhất.*

***Từ khóa***: Ngắn nhất; Greedy Best First Search; A\*

1. **Đồ thị**

***Hình 1:*** *Đồ thị thành phố*

1. **Các thuật toán được sử dụng**
   1. **Thuật toán Best First Search**

Tư tưởng của thuật toán này là việc tìm kiếm bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo có giá trị của hàm đánh giá là thấp nhất so với các nút còn lại nằm trong **hàng đợi**.

1. **Mã giả**

Best-first search {

closed list = [ ]

open list = [start node]

do{

If open list is empty then {

return no solution

}

n = heuristic best node

if n = = final node then {

return path from start to goal node

}  
Foreach direct available node do {

If node not in open and not in closed list do {

Add node to open list

Set n as his parent node

}

Delete n from open list

Add n to closed list

} While (open list is not empty)

1. **Độ phức tạp**: O(b^m)
   1. **Thuật toán A\***

Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "**đánh giá heuristic**" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của **đánh giá heuristic** này

1. **Mã giả**

A\* (điểm\_xuât\_phát, đích){

var đóng := tập rỗng

var q:= tạo\_hàng\_đợi (tạo\_đường\_đi(điểm\_xuất\_phát))

while q không phải tập rỗng {

var p:= lấy\_phần\_từ\_đầu\_tiên(p)

var x:= nút cuối cùng của p

if(x in đóng)

continue;

if(x = đích)

return p

Bổ sung x vào tập đóng

foreach y in các \_đường\_đi\_tiếp\_theo(p)

đưa\_vào\_hàng\_đợi(q,y)

}

return failure

}

1. **Độ phức tạp:** O(b^m)

**3. Ví dụ**

* **Người A xuất phát từ: Craiova (8)**
* **Người B xuất phát từ: Rimnicu (10)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Vị trí người 1 | Vị trí người 2 | Thời gian đi của người 1 | Thời gian đi của người 2 | Thời gian đợi của người 1 | Thời gian đợi của người 2 |
| Craiova | Rimnicu | 0 | 0 | 0 | 0 |
| Drobeeta | Pitesti | 120 | 97 | 0 | 23 |
| Drobeeta | Craiova | 120 | 146 | 26 | 0 |
| Drobeeta | Vilcea | 120 | 80 | 0 | 40 |
| Pitesti | Pitesti | 138 | 97 | 0 | 41 |
| Pitesti | Craiova | 138 | 146 | 8 | 0 |
| Pitesti | Vilcea | 138 | 80 | 0 | 58 |
| Rimnicu | Pitesti | 146 | 97 | 0 | 49 |
| Rimnicu | Craiova | 146 | 146 | 0 | 0 |
| Rimnicu | Vilcea | 146 | 80 | 0 | 66 |

**\*Bảng biểu:**

Giả sử người thứ 1 ở Craiova, người thứ 2 ở Rimnicu, Thuật toán A\* được sử dụng ngay tại 2 vị trí 2 người đứng, cùng tìm ra con đường ngắn nhất người cũng đi. Trước tiên, nhóm em chạy thuật toán BFS tìm đươc số đường đi ít nhất giữ 2 thành phố. Tiếp theo người thứ 1 sử dụng A\*, có thể chạy tới các vị trí Drobeta, Pitesti, Riminicu. Cùng lúc đó người thứ 2 cũng sử dụng A\* và chạy tới từng các vị trí có thể đi qua Pitesti, Craiova, Vilcea. Vì nhờ thuật toán BFS đã tìm được số đường đi ngắn nhất giữa 2 thành phố

1. **Kết luận.**

*Qua đề tài này, cho chúng ta đã thấy thuật toán* ***Greedy best first search*** *hoạt động như thế nào để giúp chúng ta tìm ra con đường ngắn nhất từ ​​nút này sang nút khác. Tuy nhiên, thuật toán này còn một số chỗ cần cải thiện, vì vậy nếu chúng ta có một số chi phí ước tính từ mỗi nút đến đích, chúng ta có thể tìm thấy con đường* ***ngắn nhất*** *nhanh hơn nhiều. Và đây là cách chúng tôi chuyển đổi thuật toán Greedy best first search thành thuật toán* ***A sao (A \*).***

**TÀI LIỆU THAM KHẢO**

**[1]** [**https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G**](https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G)

**[2]**[**https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i\_thu%E1%BA%ADt\_t%C3%ACm\_ki%E1%BA%BFm\_A\***](https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_t%C3%ACm_ki%E1%BA%BFm_A*)

**[3]** [**https://helpex.vn/article/tu-dijkstra-den-a-star-a-phan-2-thuat-toan-a-star-a-5c6b287fae03f628d053c721**](https://helpex.vn/article/tu-dijkstra-den-a-star-a-phan-2-thuat-toan-a-star-a-5c6b287fae03f628d053c721)