TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI BÁO CÁO GIỮA KỲ MÔN  
NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

*Người hướng dẫn:* **TS. BÙI THANH HÙNG**

*Người thực hiện*: **TRẦN QUỐC TÂM – 51800721**

*Lớp:* **18050402**

*Khóa:* **22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI BÁO CÁO GIỮA KỲ MÔN  
NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

*Người hướng dẫn:* **TS. BÙI THANH HÙNG**

*Người thực hiện*: **TRẦN QUỐC TÂM – 51800721**

*Lớp:* **18050402**

*Khóa:* **22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

# LỜI CẢM ƠN

Em xin gởi lời cảm ơn đến với thầy Bùi Thanh Hùng, giảng viên lý thuyết kiêm người hướng dẫn em trong quá trình làm báo cáo môn Xử lý dữ liệu lớn.

Nhờ có sự dẫn dắt từ thầy trong thời gian qua mà em đã có được rất nhiều kiến thức bổ ích về môn học này. Xin cảm ơn thầy đã hỗ trợ để em có thể hoàn thành bài báo cáo này.

Do còn ít kinh nghiệm nên bài báo cáo đôi chỗ vẫn còn sai sót, em mong có được sự góp ý của thầy để em có thể hoàn thiện hơn nữa khả năng viết báo cáo của mình sau này. Một lần nữa, em xin chân thành cảm ơn.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH  
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của giảng viên Bùi Thanh Hùng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Khóa luận/Đồ án tốt nghiệp còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Khóa luận/Đồ án tốt nghiệp của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Trần Quốc Tâm*

# PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

# TÓM TẮT

Trong thời đại công nghệ số đang ngày càng phát triển mạnh mẽ như hiện nay thì việc ứng dụng công nghệ thông tin vào cuộc sống hằng ngày dần không còn xa lạ gì với mọi người. Bởi vì công nghệ là một công cụ giúp quản lý kiểm soát một cách chính xác cao, tổ chức hiệu quả và dễ dàng tiếp cận, giảm thời gian làm việc. Từ những gì mà công nghệ thông tin có thể đem lại cho mọi người là không thể chối bỏ. Chính vì sự phát triển không ngừng nghỉ như thế mà việc ứng dụng một khái niệm không tưởng là trí tuệ nhân tạo vào cuộc sống cũng không phải là ngoại lệ.

Vậy trí tuệ nhân tạo (AI) là gì? AI (Artificial Intelligence) – là trí tuệ nhân tạo hay còn được gọi là trí tuệ thông minh. Artificial Intelligence được con người tạo ra để thể hiện trí tuệ thông qua các máy móc. Mục đích của chúng là giúp máy tính có thể tự động hóa các hành vi thông minh như con người. Ngay nay, mọi người có thể bất gặp trí tuệ nhân tạo ở bất kì đâu như trong y tế, giao dục, du lịch và giải trí, … Có thể thấy, AI có sức ảnh hưởng vô cùng lớn đến cuộc sống của loài người ở hiện tại và tương lai. Nó góp phần mở ra một kỷ nguyên văn minh mới của loài người, kỷ nguyên công nghệ.

Bài báo cáo giữa kì môn nhập môn trí tuệ nhân tạo này gồm hai chương. Chương 1 sẽ giới thiệu các thuật toán Heuristic, Best First Search, Breadth First Search và Uniform Cost Search. Riêng ở 2 giải thuật Breadth First Search và Uniform Cost Search thì mỗi phần sẽ gồm 3 phần gồm: Cấu trúc dữ liệu, thuật toán và kiểm tra đánh giá 2 giải thuật này. Chương 2 cũng sẽ giới thiệu cấu trúc dữ liệu, thuật toán và kiểm tra đánh giá của hai giải thuật khác là Greedy Best First Search và A\*.

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc100172608)

[ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG ii](#_Toc100172609)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc100172610)

[TÓM TẮT iv](#_Toc100172611)

[MỤC LỤC 1](#_Toc100172612)

[DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT 3](#_Toc100172613)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 4](#_Toc100172614)

[CHƯƠNG 1 - ISLANDS 7](#_Toc100172615)

[1.1. Heuristic 7](#_Toc100172616)

[1.2. Best First Search 8](#_Toc100172617)

[1.2.1. Best First Search là gì? 8](#_Toc100172618)

[1.2.2. Thuật toán Best First Search 9](#_Toc100172619)

[1.2.3. Các biến thể của thuật toán Best First Search 10](#_Toc100172620)

[1.3. Breadth First Search 10](#_Toc100172621)

[1.3.1. Cấu trúc dữ liệu 10](#_Toc100172622)

[1.3.2. Thuật toán giải thuật Breadth First Search. 19](#_Toc100172623)

[1.3.3. Test 25](#_Toc100172624)

[1.4. Uniform Cost Search 27](#_Toc100172625)

[1.4.1. Cấu trúc dữ liệu 27](#_Toc100172626)

[1.4.2. Thuật toán 28](#_Toc100172627)

[1.4.3. Test 35](#_Toc100172628)

[CHƯƠNG 2 – DU LỊCH Ở ROMANIA 37](#_Toc100172629)

[2.1 Greedy Best First Search 37](#_Toc100172630)

[2.1.1. Cấu trúc dữ liệu 37](#_Toc100172631)

[2.1.2. Thuật toán 37](#_Toc100172632)

[2.1.3. Test 41](#_Toc100172633)

[2.2 A\* Algorithm 42](#_Toc100172634)

[2.2.1. Cấu trúc dữ liệu 42](#_Toc100172635)

[2.2.2. Thuật toán A\* 43](#_Toc100172636)

[2.2.3. Test 50](#_Toc100172637)

[TỰ ĐÁNH GIÁ 53](#_Toc100172638)

[TÀI LIỆU THAM KHẢO 54](#_Toc100172639)

# DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

**CÁC CHỮ VIẾT TẮT**

BFS: Best First Search

AI: Artificial Intelligence

UCS: Uniform Cost Search

BFS: Breadth First Search

DFS: Depth First Search

# DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC BẢNG**

**CHƯƠNG 1:**

[Hình 1. 1: Minh họa Breadth First Search. Các con số cho biết thứ tự mà các nút được kiểm tra. 10](#_Toc100172690)

[Hình 1. 2: Mô tả bước 1 của thuật toán BFS 11](#_Toc100172691)

[Hình 1. 3: Mô tả bước 2 của thuật toán BFS 12](#_Toc100172692)

[Hình 1. 4: Mô tả bước 3 của thuật toán BFS 13](#_Toc100172693)

[Hình 1. 5: Mô tả bước 4 của thuật toán BFS 14](#_Toc100172694)

[Hình 1. 6: Mô tả bước 5 của thuật toán BFS 15](#_Toc100172695)

[Hình 1. 7: Mô tả bước 6 của thuật toán BFS 16](#_Toc100172696)

[Hình 1. 8: Mô tả bước 7 của thuật toán BFS 17](#_Toc100172697)

[Hình 1. 9: Đồ thị thuật toán giải thuật BFS 18](#_Toc100172698)

[Hình 1. 10: Ma trận 10x10 chưa được đánh số 19](#_Toc100172699)

[Hình 1. 11: Ma trận 10x10 sau khi được đánh số 20](#_Toc100172700)

[Hình 1. 12: Code hiện thực của thuật toán BFS(1) 21](#_Toc100172701)

[Hình 1. 13: Code hiện thực của thuật toán BFS(2) 22](#_Toc100172702)

[Hình 1. 14: Code hiện thực của thuật toán BFS(3) 22](#_Toc100172703)

[Hình 1. 15: Code hiện thực của thuật toán BFS(4) 23](#_Toc100172704)

[Hình 1. 16: Kết quả code hiện thực của thuật toán BFS 23](#_Toc100172705)

[Hình 1. 17: Ví dụ thuật toán Uniform Cost Search 27](#_Toc100172706)

[Hình 1. 18: Ví dụ UCS bước 1 28](#_Toc100172707)

[Hình 1. 19: Ví dụ UCS bước 2 28](#_Toc100172708)

[Hình 1. 20: Ví dụ UCS bước 3 29](#_Toc100172709)

[Hình 1. 21: Ví dụ UCS bước 4 29](#_Toc100172710)

[Hình 1. 22: Ví dụ UCS bước 5 30](#_Toc100172711)

[Hình 1. 23: Ví dụ UCS bước 6 31](#_Toc100172712)

[Hình 1. 24: Ví dụ UCS bước 7 31](#_Toc100172713)

[Hình 1. 25: Ví dụ UCS bước 8 32](#_Toc100172714)

**CHƯƠNG 2:**

[Hình 2. 1: Bản đồ du lịch ở Romania 37](#_Toc100172732)

[Hình 2. 2: Du lịch ở Romania áp dụng thuật toán GBFS – bước 1 38](#_Toc100172733)

[Hình 2. 3: Du lịch ở Romania áp dụng thuật toán GBFS – bước 2 38](#_Toc100172734)

[Hình 2. 4: Du lịch ở Romania áp dụng thuật toán GBFS – bước 3 38](#_Toc100172735)

[Hình 2. 5: Du lịch ở Romania áp dụng thuật toán GBFS – bước 4 39](#_Toc100172736)

[Hình 2. 6: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(1) 39](#_Toc100172737)

[Hình 2. 7: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(2) 40](#_Toc100172738)

[Hình 2. 8: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(3) 40](#_Toc100172739)

[Hình 2. 9: Kết quả của du lịch ở Romania áp dụng thuật toán GBFS 40](#_Toc100172740)

[Hình 2. 10: Bản đồ du lịch ở Romania 43](#_Toc100172741)

[Hình 2. 11: Du lịch ở Romania áp dụng thuật toán A\* - bước 1 44](#_Toc100172742)

[Hình 2. 12: Du lịch ở Romania áp dụng thuật toán A\* - bước 2 44](#_Toc100172743)

[Hình 2. 13: Du lịch ở Romania áp dụng thuật toán A\* - bước 3 45](#_Toc100172744)

[Hình 2. 14: Du lịch ở Romania áp dụng thuật toán A\* - bước 4 45](#_Toc100172745)

[Hình 2. 15: Du lịch ở Romania áp dụng thuật toán A\* - bước 5 46](#_Toc100172746)

[Hình 2. 16: Du lịch ở Romania áp dụng thuật toán A\* - bước 6 46](#_Toc100172747)

[Hình 2. 17: Du lịch ở Romania áp dụng thuật toán A\* - bước 7 47](#_Toc100172748)

[Hình 2. 18: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(1) 48](#_Toc100172749)

[Hình 2. 19: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(2) 48](#_Toc100172750)

[Hình 2. 20: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(3) 49](#_Toc100172751)

[Hình 2. 21: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(4) 49](#_Toc100172752)

[Hình 2. 22: Kết quả du lịch ở Romania áp dụng thuật toán A\* 49](#_Toc100172753)

**DANH MỤC HÌNH**

[Bảng 1. 1: So sánh giải thuật Breadth First Search và Depth First Search 26](#_Toc100173080)

[Bảng 2. 1: Tự đánh giá quá trình làm bài 53](#_Toc100172812)

# ISLANDS

1. Heuristic

Heuristic là hàm dựa trên kinh nghiệm để giải quyết vấn đề, học hỏi hay khám phá nhằm đưa ra một giải pháp mà không được đảm bảo là tối ưu. Với việc nghiên cứu khảo sát không có tính thực tế, các phương pháp heuristic được dùng nhằm tăng nhanh quá trình tìm kiếm với các giải pháp hợp lý thông qua các suy nghĩ rút gọn để giảm bớt việc nhận thức vấn đề khi đưa ra quyết định. Ví dụ của phương pháp này bao gồm sử dụng một luật ngón tay cái, giả thuyết, phán đoán trực giác, khuôn mẫu hay nhận thức thông thường.

Hàm Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau:

* Thường tìm được lời giải tốt (nhưng không chắc là lời giải tốt nhất)
* Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.
* Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Một số nguyên lý cơ sở được dựa vào để xây dựng một hàm Heuristic là:

* Nguyên lý vét cạn thông minh: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.
* Nguyên lý tham lam (Greedy): Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.
* Nguyên lý thứ tự: Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.

Tuy nhiên, phương pháp heuristic có thể không phải lúc nào cũng đưa ra giải pháp tốt nhất, nhưng nó đảm bảo tìm ra giải pháp tốt trong thời gian hợp lý. Hàm heuristic ước tính mức độ gần của một trạng thái với mục tiêu. Nó được đại diện bởi h (n), và nó tính toán chi phí của một con đường tối ưu giữa các cặp trạng thái. Giá trị của hàm heuristic luôn dương. Sự chấp nhận của hàm heuristic được đưa ra như sau:

***h (n) <= h \* (n)***

Với:

* h (n): là chi phí Heuristic
* h \* (n): là chi phí ước tính

Chính vì lẽ đó mà chi phí Heuristic phải nhỏ hơn hoặc bằng chi phí ước tính. Tìm kiếm theo phương pháp heuristic thuần túy là dạng đơn giản nhất của các thuật toán tìm kiếm heuristic. Nó mở rộng các nút dựa trên giá trị Heuristic “h (n)” của chúng. Đồng thời, duy trì hai danh sách, danh sách “MỞ” và “ĐÓNG”. Trong danh sách “ĐÃ ĐÓNG”, phương pháp này đặt các nút đã được mở rộng và trong danh sách MỞ, nó đặt các nút chưa được mở rộng. Trên mỗi lần lặp, mỗi nút n có giá trị Heuristic thấp nhất được mở rộng và tạo ra tất cả các nút kế tiếp của nó và n được đặt vào danh sách đóng.

1. Best First Search

1.2.1. Best First Search là gì?

Nếu coi việc tìm kiếm là một dạng duyệt trong biểu đồ, thì một thuật toán tìm kiếm không có hiểu biết sẽ đi ngang một cách mù quáng đến nút tiếp theo theo một cách nhất định mà không tính đến chi phí liên quan đến bước đó. Đối với một tìm kiếm được thông báo như Tìm kiếm đầu tiên tốt nhất, thì mặt khác sẽ sử dụng một hàm đánh giá để quyết định xem nút nào trong số các nút có sẵn khác nhau là hứa hẹn nhất (hoặc 'TỐT NHẤT') trước khi chuyển đến nút đó.

Best First Search sử dụng khái niệm hàng đợi “ưu tiên” và tìm kiếm theo phương pháp heuristic. Để tìm kiếm không gian biểu đồ, phương pháp BFS sử dụng hai danh sách để theo dõi chuyển tải. Danh sách “Mở” theo dõi các nút "ngay lập tức" hiện có để duyệt và danh sách "ĐÃ ĐÓNG" theo dõi các nút đã được duyệt.

Tìm kiếm ưu tiên nhất (Best First Search) sử dụng Heuristic theo cách tương tự như leo đồi. Sự khác biệt là với tìm kiếm ưu tiên nhất, toàn bộ hàng đợi được sắp xếp sau các đường dẫn mới đã được thêm vào nó, thay vì thêm một tập hợp các đường dẫn được sắp xếp. Theo thuật ngữ thực tế, điều này có nghĩa là tìm kiếm ưu tiên nhất đi theo đường dẫn tốt nhất có sẵn từ cây hiện tại (được phát triển một phần), thay vì luôn tuân theo cách tiếp cận theo phong cách chiều sâu.

1.2.2. Thuật toán Best First Search

Các bước thực hiện thuật toán Best First Search:

1. Tạo 2 danh sách trống: MỞ và ĐÓNG CỬA
2. Bắt đầu từ nút ban đầu (giả sử N) và đưa nó vào danh sách MỞ 'có thứ tự'
3. Lặp lại các bước tiếp theo cho đến khi đạt được nút GOAL
4. Nếu danh sách OPEN trống, sau đó EXIT vòng lặp trả về 'False'
5. Chọn nút đầu tiên / trên cùng (giả sử N) trong danh sách MỞ và di chuyển nó vào danh sách ĐÃ ĐÓNG CỬA.
6. Đồng thời nắm bắt thông tin của nút cha
7. Nếu N là một nút MỤC TIÊU, thì hãy di chuyển nút đó vào danh sách Đã đóng và thoát khỏi vòng lặp trả về ‘Đúng’. Giải pháp có thể được tìm thấy bằng cách theo dõi đường dẫn
8. Nếu N không phải là nút MỤC TIÊU, hãy mở rộng nút N để tạo các nút tiếp theo ‘ngay lập tức’ được liên kết với nút N và thêm tất cả những nút đó vào danh sách MỞ
9. Sắp xếp lại thứ tự các nút trong danh sách MỞ theo thứ tự tăng dần theo một hàm đánh giá f (n)

Thuật toán này sẽ đi qua đường ngắn nhất đầu tiên trong hàng đợi. Độ phức tạp thời gian của thuật toán được cho bởi O (n \* logn).

1.2.3. Các biến thể của thuật toán Best First Search

Thuật toán Best First Search có hai biến thể là Greedy Best First Search và A\* Best First Search. Sự khác biệt duy nhất giữa Greedy BFS và A \* BFS là ở chức năng đánh giá. Đối với Greedy BFS, hàm đánh giá là f (n) = h (n) trong khi đối với A \*, hàm đánh giá là f (n) = g (n) + h (n).

1. Breadth First Search

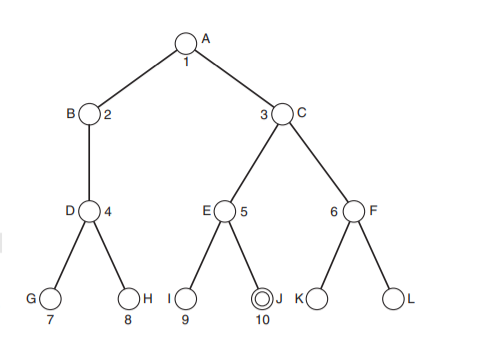
1.3.1. Cấu trúc dữ liệu

BFS là viết tắt của Breadth First Search là một thuật toán duyệt đồ thị bắt đầu duyệt đồ thị từ nút gốc và khám phá tất cả các nút lân cận. Sau đó, nó chọn nút gần nhất và khám phá tất cả các nút chưa được khám phá. Trong khi sử dụng BFS để truyền tải, bất kỳ nút nào trong đồ thị có thể được coi là nút gốc. Trong BFS, một đỉnh được chọn tại một thời điểm khi nó được truy cập và đánh dấu sau đó các đỉnh lân cận của nó được truy cập và lưu trữ trong hàng đợi.

Có nhiều cách để duyệt qua đồ thị, nhưng trong số đó, BFS là cách tiếp cận được sử dụng phổ biến nhất. Nó là một thuật toán đệ quy để tìm kiếm tất cả các đỉnh của cấu trúc dữ liệu dạng cây hoặc đồ thị. BFS đặt mọi đỉnh của biểu đồ thành hai loại - được truy cập và không được truy cập. Nó chọn một nút duy nhất trong biểu đồ và sau đó, thăm tất cả các nút liền kề với nút đã chọn.

Các ứng dụng của giải thuật Breadth First Search:

* BFS có thể được sử dụng để tìm các vị trí lân cận từ một vị trí nguồn nhất định.
* Trong mạng ngang hàng, thuật toán BFS có thể được sử dụng như một phương pháp duyệt để tìm tất cả các nút lân cận. Hầu hết các ứng dụng khách torrent, chẳng hạn như BitTorrent, uTorrent, v.v. sử dụng quy trình này để tìm "hạt giống" và "đồng nghiệp" trong mạng.
* BFS có thể được sử dụng trong trình thu thập dữ liệu web để tạo chỉ mục trang web. Nó là một trong những thuật toán chính có thể được sử dụng để lập chỉ mục các trang web. Nó bắt đầu đi ngang từ trang nguồn và đi theo các liên kết được liên kết với trang. Ở đây, mọi trang web được coi là một nút trong biểu đồ.
* BFS được sử dụng để xác định đường đi ngắn nhất và cây bao trùm tối thiểu.
* BFS cũng được sử dụng trong kỹ thuật của Cheney để nhân bản bộ sưu tập rác.
* Nó có thể được sử dụng trong phương pháp ford - Fulkerson để tính lưu lượng tối đa trong mạng luồng.

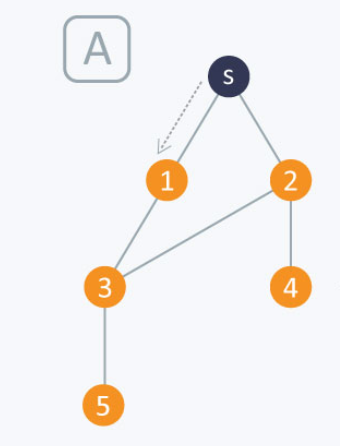


Hình 1. 1: Minh họa Breadth First Search. Các con số cho biết thứ tự mà các nút được kiểm tra.

Hiểu đơn giản thì giải thuật Breadth First Search sẽ duyệt theo cơ chế sau:

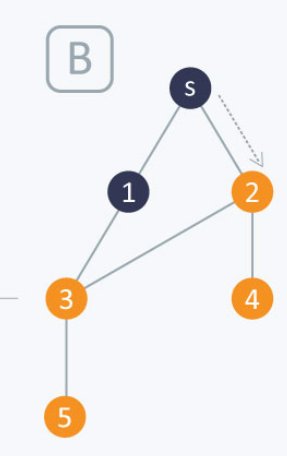
1. Đầu tiên di chuyển theo chiều ngang và truy cập tất cả các nút của lớp hiện tại
2. Di chuyển đến lớp tiếp theo

Để hiểu hơn về cách thức hoạt động của giải thuật Breadth First Search thì ta hay tìm hiểu kỹ hơn thông qua quá trình duyệt của thuật toán này.



Hình 1. 2: Mô tả bước 1 của thuật toán BFS

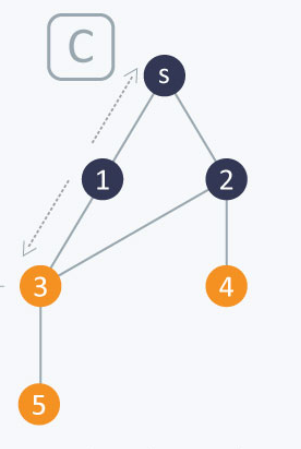
1. Việc duyệt sẽ bắt đầu từ nút nguồn và đẩy s vào hàng đợi. s sẽ được đánh dấu là 'đã ghé thăm'.



Hình 1. 3: Mô tả bước 2 của thuật toán BFS

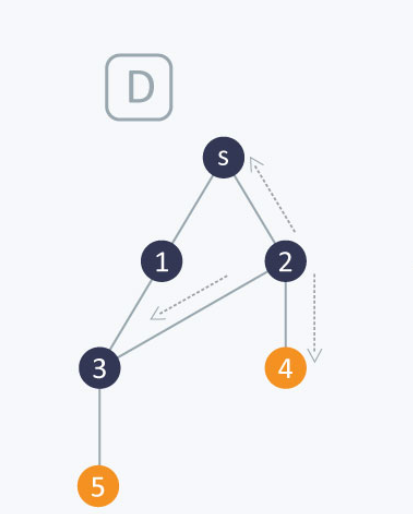
1. Lần lặp đầu tiên
   * s sẽ được xuất hiện từ hàng đợi
   * Các vùng lân cận của s, tức là 1 và 2 sẽ được chuyển qua
   * 1 và 2, chưa được duyệt trước đó, được duyệt qua. Họ sẽ được:

* Được đẩy vào hàng đợi
* 1 và 2 sẽ được đánh dấu là đã ghé thăm



Hình 1. 4: Mô tả bước 3 của thuật toán BFS

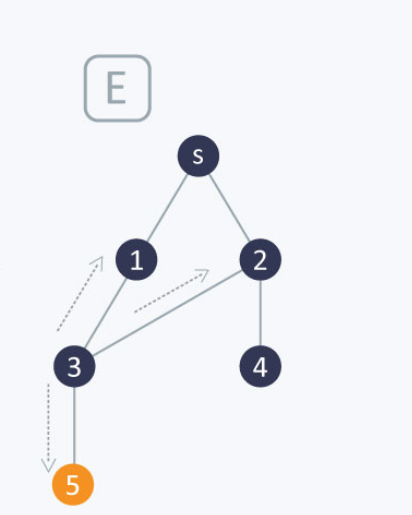
1. Lần lặp thứ hai
   * 1 được xuất hiện từ hàng đợi
   * Hàng xóm của 1 tức là 3 và 3 bị đi ngang
   * s bị bỏ qua vì nó được đánh dấu là 'đã ghé thăm'
   * 3, chưa được duyệt trước đó, được duyệt qua. Nó là:
     + Được đẩy vào hàng đợi
     + Được đánh dấu là đã ghé thăm



Hình 1. 5: Mô tả bước 4 của thuật toán BFS

1. Lần lặp thứ ba
   * 2 được xuất hiện từ hàng đợi
   * Các vùng lân cận của 2 tức là 3, 4 được đi ngang
   * 3 và s bị bỏ qua vì chúng được đánh dấu là 'đã ghé thăm'
   * 4, chưa được duyệt trước đó, được duyệt qua. Nó là:

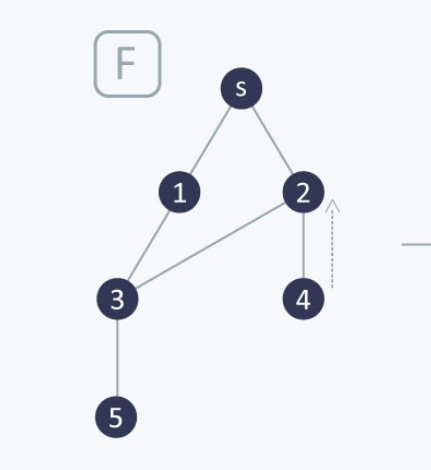
* Được đẩy vào hàng đợi
* Được đánh dấu là đã ghé thăm



Hình 1. 6: Mô tả bước 5 của thuật toán BFS

1. Lần lặp thứ tư
   * 3 được xuất hiện từ hàng đợi
   * Các vùng lân cận của 3 tức là 1, 2 và 5 được cắt ngang
   * 1 và 2 bị bỏ qua vì chúng được đánh dấu là 'đã ghé thăm'
   * 5, chưa được duyệt trước đó, được duyệt qua. Nó là:

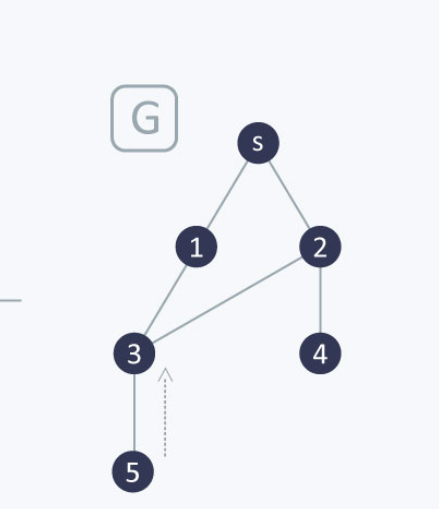
* Được đẩy vào hàng đợi
* Được đánh dấu là đã ghé thăm



Hình 1. 7: Mô tả bước 6 của thuật toán BFS

1. Lần lặp thứ năm

* 4 sẽ được xuất hiện từ hàng đợi
* Hàng xóm của 4 tức là 2 được đi ngang
* 2 bị bỏ qua vì nó đã được đánh dấu là 'đã ghé thăm'



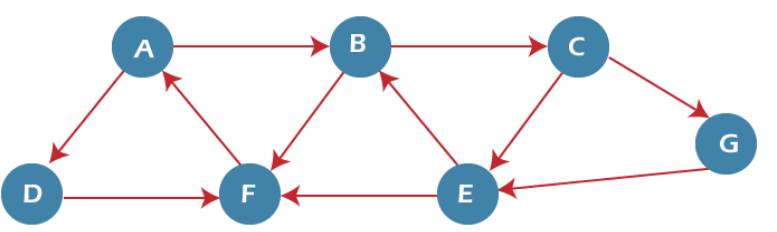
Hình 1. 8: Mô tả bước 7 của thuật toán BFS

1. Lần lặp thứ sáu

* 5 được xuất hiện từ hàng đợi
* Hàng xóm của 5 tức là 3 được đi ngang
* 3 bị bỏ qua vì nó đã được đánh dấu là 'đã ghé thăm'

1.3.2. Thuật toán giải thuật Breadth First Search.

Các bước thực hiện thuật toán Breadth First Search để khám phá một đồ thị như sau:



Hình 1. 9: Đồ thị thuật toán giải thuật BFS

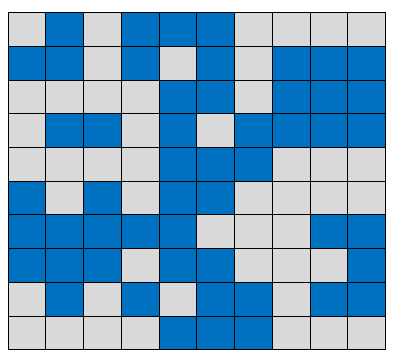
* Bước 1: ĐẶT TRẠNG THÁI = 1 (trạng thái sẵn sàng) cho mỗi nút trong G
* Bước 2: Xếp hàng nút bắt đầu A và đặt STATUS = 2 (trạng thái chờ) của nó
* Bước 3: Lặp lại các bước 4 và 5 cho đến khi QUEUE trống
* Bước 4: Dequeue một nút N. Xử lý nó và đặt STATUS = 3 (trạng thái đã xử lý) của nó.
* Bước 5: Xếp hàng tất cả các lân cận của N đang ở trạng thái sẵn sàng (có STATUS = 1) và đặt
* STATUS = 2 (trạng thái chờ đợi)
* [KẾT THÚC LOOP]
* Bước 6: THOÁT

Một đồ thị trong đó tất cả các đỉnh được kết nối với nhau có đúng một thành phần được kết nối, bao gồm toàn bộ đồ thị. Đồ thị chỉ có một thành phần được kết nối như vậy được gọi là Đồ thị được kết nối mạnh mẽ (Strongly Connected Graph).

Đối với việc sử dụng giải thuật Breadth First Search cho việc tìm kiếm số hòn đảo trong một ma trận thì chúng ta sẽ tiến hành thiết lập một ma trận nhị phân trong đó 0 đại diện nước và 1 đại diện đất liền, và các ô vuông được được kết nối với nhau tạo thành một hòn đảo. Trong mỗi lệnh gọi BFS (), một thành phần hoặc một biểu đồ con được truy cập. Chúng ta sẽ gọi BFS trên thành phần chưa được truy cập tiếp theo. Số lượng cuộc gọi đến BFS () cho biết số lượng các thành phần được kết nối. Từ đó sẽ hình thành nên hòn đảo và tiến hành đếm tổng số hòn đảo đang có trên ma trận đó.

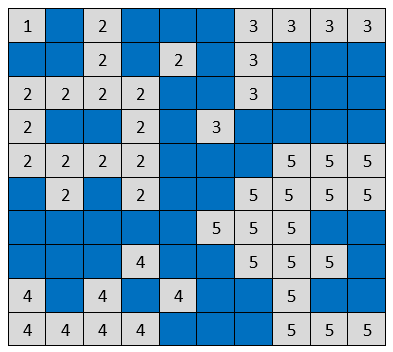
Mỗi một ô trong một ma trận 2D có thể được kết nối với 8 “hàng xóm”. Vì vậy không giống như Breadth First Search tiêu chuẩn thường thấy bằng việc sử lý tất cả các đỉnh liền kề mà ở đây sẽ cần sử lý 8 thành phần lân cận xung quanh.

Như ví dụ trong hình, chúng ta có thể thấy rằng:



Hình 1. 10: Ma trận 10x10 chưa được đánh số

Hình ảnh trên làm nổi bật nước có màu xanh và đất có màu xám trong ma trận 10 x 10. Có tổng cộng năm hòn đảo hiện diện trong ma trận trên. Chúng được đánh dấu bằng các số 1-5 trong hình dưới đây.



Hình 1. 11: Ma trận 10x10 sau khi được đánh số

Giải pháp được lấy cảm hứng từ việc tìm tổng số các thành phần được kết nối trong một bài toán đồ thị. Ý tưởng là bắt đầu tìm kiếm theo chiều rộng (BFS) từ mỗi nút chưa được xử lý và tăng số lượng đảo. Mỗi lần truyền BFS sẽ đánh dấu tất cả các ô tạo thành một đảo là đã được xử lý. Vì vậy, vấn đề giảm xuống việc tìm tổng số cuộc gọi BFS.

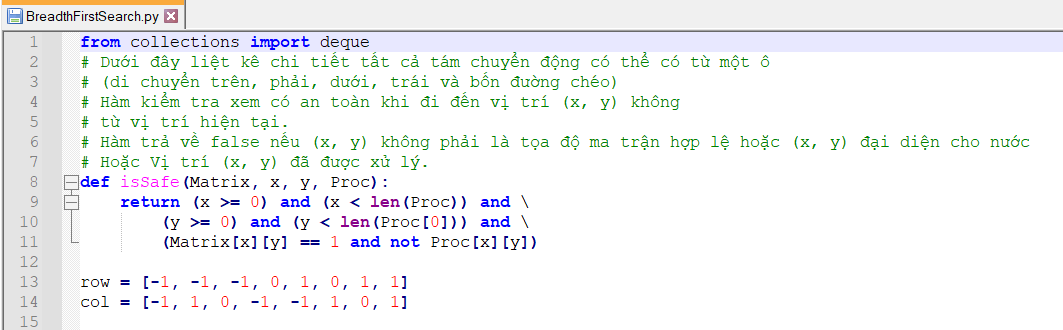
Trong mỗi lần truyền BFS, hãy bắt đầu bằng cách tạo một hàng đợi trống. Sau đó xếp ô bắt đầu và đánh dấu là đã xử lý. Tiếp theo bằng cách dequeue nút phía trước, xử lý tất cả tám ô liền kề của ô hiện tại và xếp hàng mỗi ô hợp lệ, đó là đất. Lặp lại quá trình này cho đến khi hàng đợi không còn trống.

Chúng ta có thể tìm thấy tất cả các vị trí có thể mà chúng ta có thể di chuyển đến từ vị trí đã cho bằng cách sử dụng mảng lưu trữ vị trí tương đối của chuyển động từ bất kỳ vị trí nào. Ví dụ: nếu vị trí hiện tại là (x, y), chúng ta có thể di chuyển đến (x + row[k], y + col[k]) cho 0 <= k <= 7 bằng cách sử dụng các mảng sau:

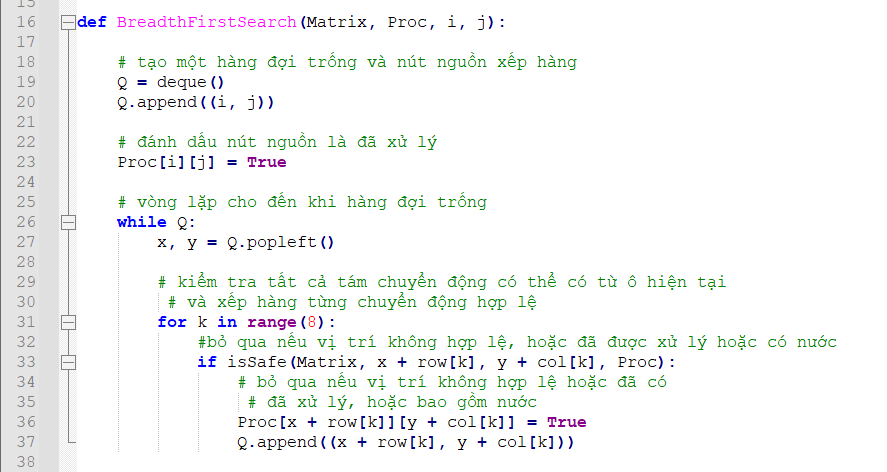
* int row [] = { -1, -1, -1, 0, 0, 1, 1, 1}
* int col [] = { -1, 0, 1, -1, 1, -1, 0, 1}

Vì vậy, từ vị trí (x, y), chúng ta có thể chuyển sang:

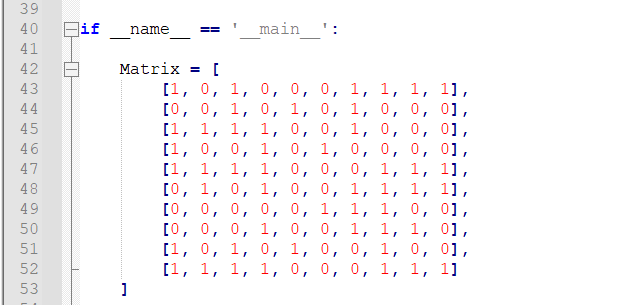
* (x - 1, y - 1)
* (x – 1, y)
* (x – 1, y + 1)
* (x, y – 1)
* (x, y + 1)
* (x + 1, y - 1)
* (x + 1, y)
* (x + 1, y + 1)
* Thực hiện thuật toán Breadth First Search



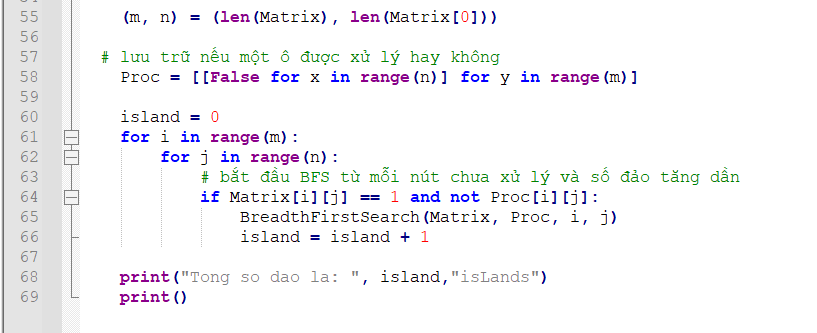
Hình 1. 12: Code hiện thực của thuật toán BFS(1)



Hình 1. 13: Code hiện thực của thuật toán BFS(2)



Hình 1. 14: Code hiện thực của thuật toán BFS(3)



Hình 1. 15: Code hiện thực của thuật toán BFS(4)



Hình 1. 16: Kết quả code hiện thực của thuật toán BFS

1.3.3. Test

**Giải thuật Breadth First Search và giải thuật Depth First Search khác nhau như thế nào?**

|  |  |
| --- | --- |
| Breadth First Search | Depth First Search |
| BFS (Tìm kiếm đầu tiên theo chiều rộng) sử dụng cấu trúc dữ liệu Queue để tìm đường đi ngắn nhất. | DFS (Tìm kiếm đầu tiên theo chiều sâu) sử dụng cấu trúc dữ liệu Stack. |
| BFS có thể được sử dụng để tìm đường đi ngắn nhất nguồn duy nhất trong một đồ thị không trọng số, bởi vì trong BFS, chúng ta đạt đến một đỉnh có số cạnh tối thiểu từ một đỉnh nguồn. | Trong DFS, chúng ta có thể đi qua nhiều cạnh hơn để đến một đỉnh đích từ một nguồn. |
| BFS thích hợp hơn để tìm kiếm các đỉnh gần với nguồn nhất định. | DFS phù hợp hơn khi có các giải pháp xa nguồn. |
| BFS xem xét tất cả những người hàng xóm trước tiên và do đó không thích hợp để ra quyết định cây được sử dụng trong trò chơi hoặc câu đố. | DFS phù hợp hơn cho các vấn đề về trò chơi hoặc giải đố. Chúng tôi đưa ra quyết định, sau đó khám phá tất cả các con đường thông qua quyết định này. Và nếu quyết định này dẫn đến tình thế chiến thắng, chúng tôi dừng lại. |
| Độ phức tạp về thời gian của BFS là O (V + E) khi Danh sách gần kề được sử dụng và O (V ^ 2) khi Ma trận kề được sử dụng, trong đó V là đỉnh và E là cạnh. | Độ phức tạp về thời gian của DFS cũng là O (V + E) khi sử dụng Danh sách gần kề và O (V ^ 2) khi sử dụng Ma trận kề, trong đó V là đỉnh và E là cạnh. |
| Với BFS thì “anh, chị” sẽ được “thăm” trước “em nhỏ”. | Với DFS thì các “em nhỏ” sẽ được thăm trước “anh chị”. |

Bảng 1. 1: So sánh giải thuật Breadth First Search và Depth First Search

Tìm kiếm theo chiều rộng (BFS) là một phương pháp tốt hơn nhiều để sử dụng trong các tình huống mà cây có thể có các đường dẫn rất sâu và đặc biệt là khi nút mục tiêu nằm ở phần nông hơn của cây. Nhưng nó lại không hoạt động tốt khi hệ số phân nhánh của cây rất cao, chẳng hạn như khi kiểm tra cây trò chơi cho các trò chơi như cờ vây hoặc cờ vua.

Breadth First Search là một ý tưởng tồi trong các cây mà tất cả các đường dẫn đều dẫn đến một nút mục tiêu có các đường dẫn có độ dài tương tự nhau. Trong những tình huống như thế này, Depth First Search sẽ hoạt động tốt hơn nhiều vì giải thuật này sẽ xác định được một nút mục tiêu khi đến cuối đường dẫn đầu tiên mà nó đã kiểm tra.

Tìm kiếm theo chiều sâu (DFS) thường dễ triển khai hơn tìm kiếm theo chiều rộng (BFS) và thường yêu cầu sử dụng ít bộ nhớ hơn vì nó chỉ cần lưu trữ thông tin về đường dẫn mà giải thuật này hiện đang khám phá, trong khi tìm kiếm theo chiều rộng cần lưu trữ thông tin về tất cả các đường dẫn đạt đến độ sâu hiện tại. Đây là một trong những lý do chính mà tìm kiếm theo chiều sâu được sử dụng rộng rãi để giải quyết các vấn đề máy tính hàng ngày.

Tóm lại, giải thuật Breadth First Seach có các đặc điểm sau:

* Độ phức tạp về thời gian: Tương đương với số lượng nút được duyệt trong BFS cho đến khi có giải pháp nông nhất.
* Độ phức tạp của không gian: Tương đương với mức độ lớn mà phần rìa có thể nhận được.
* Tính đầy đủ: BFS đã hoàn thành, nghĩa là đối với một cây tìm kiếm nhất định, BFS sẽ đưa ra giải pháp nếu nó tồn tại.
* Tính tối ưu: BFS là tối ưu miễn là chi phí của tất cả các cạnh bằng nhau.

1. Uniform Cost Search

1.4.1. Cấu trúc dữ liệu

Tìm kiếm theo chi phí thống nhất (Uniform Cost Search) là một thuật toán tìm kiếm được sử dụng để duyệt qua một cây hoặc đồ thị có trọng số. Thuật toán này phát huy tác dụng khi có sẵn một chi phí khác nhau cho mỗi cạnh. Mục tiêu chính của tìm kiếm chi phí thống nhất là tìm đường dẫn đến nút mục tiêu có chi phí tích lũy thấp nhất.

Tìm kiếm chi phí thống nhất mở rộng các nút theo chi phí đường dẫn của chúng tạo thành nút gốc. Nó có thể được sử dụng để giải quyết bất kỳ biểu đồ/ cây nào có nhu cầu về chi phí tối ưu. Thuật toán tìm kiếm chi phí thống nhất được thực hiện bởi hàng đợi ưu tiên. Nó ưu tiên tối đa cho chi phí tích lũy thấp nhất. Tìm kiếm chi phí thống nhất tương đương với thuật toán Best First Search nếu chi phí đường dẫn của tất cả các cạnh là như nhau.

1.4.2. Thuật toán

Các bước thực hiện thuật toán Uniform Cost Search:

Bước 1: Chèn RootNode vào hàng đợi.

Bước 2: Lặp lại cho đến khi hàng đợi không trống.

Bước 3: Xóa phần tử tiếp theo có mức ưu tiên cao nhất khỏi hàng đợi.

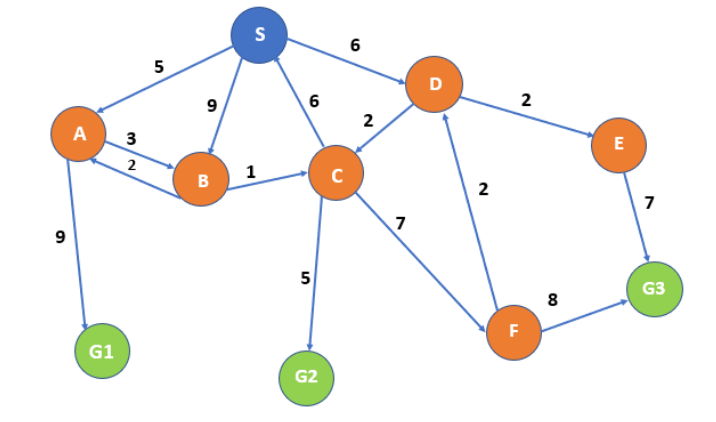
Bước 4: Nếu nút là một nút đích thì in chi phí và đường dẫn và thoát, nếu không chèn tất cả các phần tử con của các phần tử đã loại bỏ vào hàng đợi với chi phí tích lũy của chúng làm mức độ ưu tiên của chúng.

Ở đây rootNode là nút bắt đầu cho đường dẫn và một hàng đợi ưu tiên đang được duy trì để duy trì đường dẫn với chi phí thấp nhất được chọn cho lần truyền tiếp theo. Trong trường hợp 2 đường dẫn có cùng chi phí truyền tải, các nút được coi là theo thứ tự bảng chữ cái.

* **Ví dụ về thuật toán Uniform Cost Search:**

Hãy xem xét ví dụ dưới đây, nơi chúng ta cần đến bất kỳ một trong các nút đích {G1, G2, G3} bắt đầu từ nút S. Nút {A, B, C, D, E và F} là các nút trung gian. Động cơ của chúng tôi là tìm đường đi từ S đến bất kỳ trạng thái đích nào với chi phí tích lũy ít nhất.

Mỗi cạnh có hướng đại diện cho hướng di chuyển được phép qua con đường đó và việc dán nhãn của nó thể hiện chi phí của một lần di chuyển qua con đường đó. Do đó, chi phí chung của con đường là tổng của tất cả các con đường.

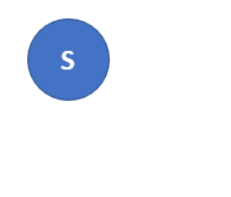


Hình 1. 17: Ví dụ thuật toán Uniform Cost Search

*Ví dụ: Đường đi từ S đến G1- {S==> A ==> G1} có chi phí là SA + AG1 = 5 + 9 = 14*

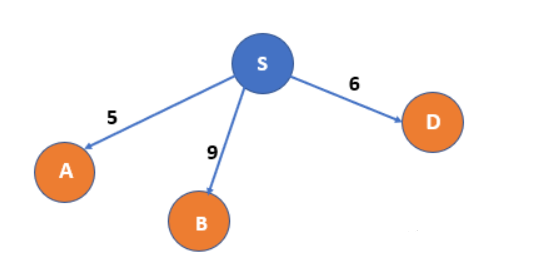
Ở đây chúng ta sẽ duy trì một hàng đợi ưu tiên giống như Best First Search với chi phí của đường dẫn là ưu tiên của nó, thấp hơn chi phí cao hơn là ưu tiên.

Bước 1: Bắt đầu với nút bắt đầu và kiểm tra xem chúng tôi đã đến được bất kỳ nút đích nào chưa, tức là không, do đó tiếp tục.



Hình 1. 18: Ví dụ UCS bước 1

Bước 2: Tiếp cận tất cả các nút có thể đạt được từ S là A, B, D. Vì nút S đã được truy cập do đó được thêm vào Danh sách đã truy cập. Bây giờ chọn con đường rẻ nhất trước tiên để mở rộng hơn nữa, tức là A = 5.

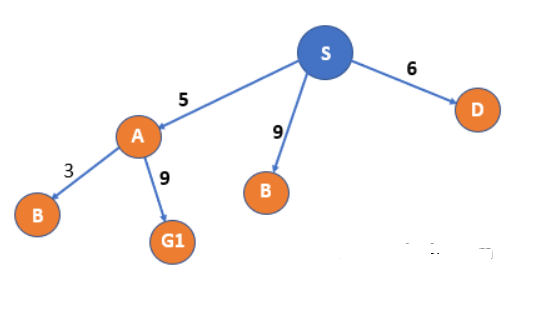


Hình 1. 19: Ví dụ UCS bước 2

Danh sách đã “ghé thăm”: S

Bước 3: Có thể đến nút B và G1 từ A và vì nút A được truy cập do đó chuyển sang danh sách đã truy cập.

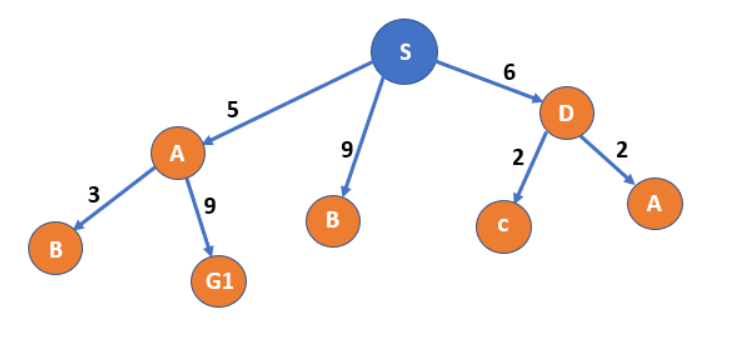
Vì đã đạt đến G1 nhưng để có giải pháp tối ưu, cần xem xét mọi trường hợp có thể xảy ra; do đó sẽ mở rộng con đường rẻ nhất tiếp theo, tức là S==> D.



Hình 1. 20: Ví dụ UCS bước 3

Danh sách đã “ghé thăm”: S, A

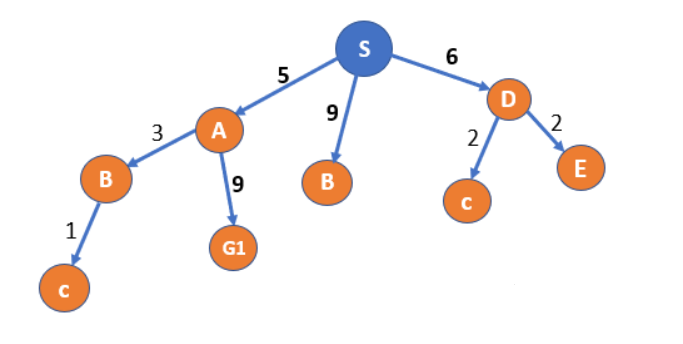
Bước 4: Bây giờ nút D đã được truy cập do đó nó sẽ chuyển sang danh sách đã truy cập và bây giờ vì có ba đường dẫn với cùng chi phí, cho nên sẽ chọn theo thứ tự bảng chữ cái do đó sẽ mở rộng nút B



Hình 1. 21: Ví dụ UCS bước 4

Danh sách đã “ghé thăm”: S, A, D

Bước 5: Từ B chỉ có thể đến nút C. Bây giờ đường dẫn có trọng số tối thiểu là S 🡺 D 🡺 C, tức là 8. Do đó mở rộng C. Và B bây giờ đã đến thăm nút.

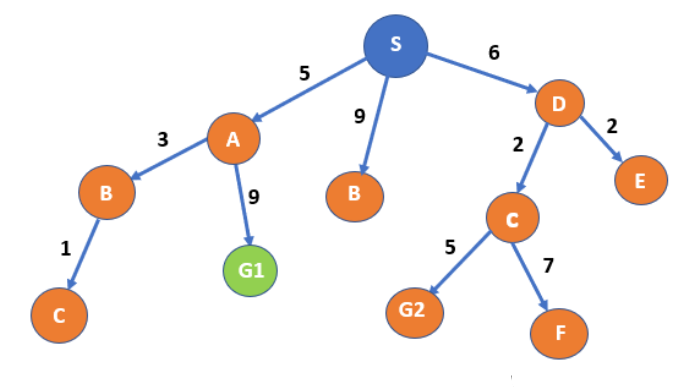


Hình 1. 22: Ví dụ UCS bước 5

Danh sách đã “ghé thăm”: S, A, D, B

Bước 6: Từ C đến được nút G2 và nút F có trọng số lần lượt là 5 và 7. Vì S hiện diện trong danh sách đã thăm nên không cần xem xét đường dẫn C==> S.

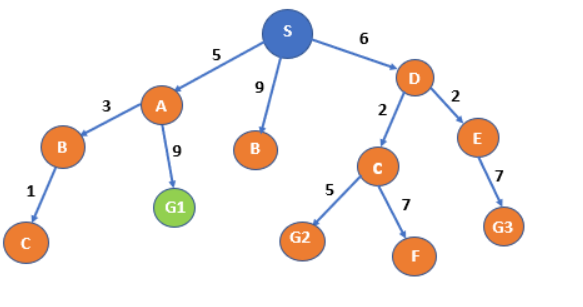
Bây giờ C sẽ nhập danh sách đã thăm. Bây giờ nút tiếp theo có tổng đường đi nhỏ nhất là S 🡺 D 🡺 E, tức là 8. Do đó chúng ta sẽ mở rộng E.



Hình 1. 23: Ví dụ UCS bước 6

Danh sách đã “ghé thăm”: S, A, D, B, C

Bước 7: Từ E chỉ có thể đến được G3. E sẽ chuyển sang danh sách đã thăm.



Hình 1. 24: Ví dụ UCS bước 7

Danh sách đã “ghé thăm”: S, A, D, B, C, E

Bước 8 - Cuối cùng, chúng ta có 6 đường dẫn hoạt động

* S 🡺 B - B nằm trong danh sách đã thăm; do đó sẽ được đánh dấu là một ngõ cụt.
* Tên cho S 🡺 A 🡺 B 🡺 C (C đã được truy cập vì vậy được coi là một ngõ cụt.)

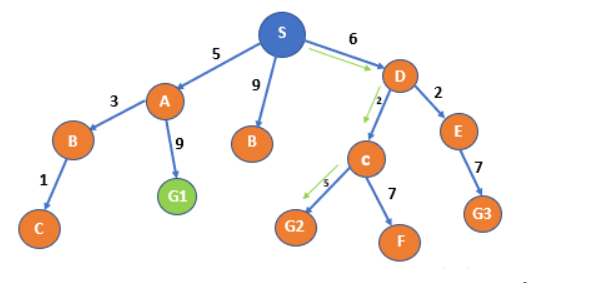
Trong số còn lại:

* S 🡺 A 🡺 G1
* S 🡺 D 🡺 C 🡺 G2
* S 🡺 D 🡺 C 🡺 F
* S 🡺 D 🡺 E 🡺 G3

Tối thiểu là S 🡺 D 🡺 C 🡺 G2

Và ngoài ra, G2 là một trong những nút đích. Vì vậy, đã tìm thấy con đường cần tìm.

Bằng cách này, có thể tìm đường dẫn với chi phí tích lũy tối thiểu từ nút bắt đầu đến nút kết thúc - S-> D-> C-> G2 với tổng chi phí là 13 (được đánh dấu bằng màu xanh lá cây).



Hình 1. 25: Ví dụ UCS bước 8

Danh sách đã “ghé thăm”: S, A, D, B, C, E

* Thực hiện thuật toán Uniform Cost Search:

1.4.3. Test

1.4.3.1. Những thuận lợi và các mặt hạn chế mà thuật toán Uniform Cost Search có:

***Thuận lợi:***

* Uniform Cost Search giúp tìm ra đường đi với chi phí tích lũy thấp nhất bên trong một đồ thị có trọng số có chi phí khác nhau được liên kết với mỗi cạnh của nó từ nút gốc đến nút đích.
* Uniform Cost Search được coi là một giải pháp tối ưu vì ở mỗi trạng thái, con đường ít nhất được coi là phải đi theo.

***Hạn chế:***

* Danh sách mở được yêu cầu phải được sắp xếp vì các ưu tiên trong hàng đợi ưu tiên cần được duy trì.
* Bộ nhớ được yêu cầu lớn theo cấp số nhân.
* Thuật toán có thể bị mắc kẹt trong một vòng lặp vô hạn vì nó xem xét mọi con đường có thể đi từ nút gốc đến nút đích.

1.4.3.2. Sự khác nhau giữa Unifrom Cost Search và Best First Search.

Sự khác biệt là ở hàm heuristic.

Unifrom Cost Search là tìm kiếm không có thông tin: giải thuật này không sử dụng bất kỳ kiến ​​thức miền nào. UCS mở rộng nút chi phí thấp nhất và nó làm như vậy theo mọi hướng vì không có thông tin về mục tiêu được cung cấp.

Uniform Cost Search có thể được xem như một hàm f (n) = g (n) trong đó g (n) là chi phí đường dẫn (bản thân "chi phí đường dẫn" là một hàm chỉ định chi phí số cho một đường dẫn liên quan đến thước đo hiệu suất, ví dụ: khoảng cách tính bằng km, hoặc số lần di chuyển, v.v.). Thuật toán này chỉ đơn giản là một chi phí để đạt được nút n.

Best First Search là tìm kiếm có thông tin: giải thuật này sử dụng hàm heuristic để ước tính mức độ gần nhất của trạng thái hiện tại với mục tiêu (chúng ta có đang tiến gần đến mục tiêu không?). Do đó, hàm chi phí của chúng ta f (n) = g (n) được kết hợp với chi phí để đi từ n đến mục tiêu, h (n) (hàm heuristic ước tính chi phí đó) cho chúng ta f (n) = g (n) + h (n).

Tóm lại, cả hai phương pháp đều có điểm chung là có danh sách các nút mở rộng, nhưng Best First Search sẽ cố gắng giảm thiểu số lượng nút mở rộng đó (chi phí đường dẫn + hàm heuristic).

1.4.3.3. Đặc điểm của thuật toán Uniform Cost Search.

1. Tính hoàn chỉnh: Uniform Cost Search đã hoàn tất, chẳng hạn như nếu có giải pháp, UCS sẽ tìm ra giải pháp đó.
2. Độ phức tạp về thời gian:

* Gọi C \* là chi phí của giải pháp tối ưu (Cost of the optimal solution) và ε là từng bước để tiến gần hơn đến nút mục tiêu. Khi đó số bước là = C \* / ε + 1. Ở đây, đã lấy +1, khi bắt đầu từ trạng thái 0 và kết thúc đến C \* / ε.
* Do đó, độ phức tạp thời gian trong trường hợp xấu nhất của Uniform Cost Search làO (b1 + [C \* / ε]).

1. Không gian phức tạp: Logic tương tự là đối với độ phức tạp không gian, vì vậy, độ phức tạp không gian trong trường hợp xấu nhất của Tìm kiếm theo chi phí thống nhất là O (b1 + [C \* / ε]).
2. Tối ưu: Tìm kiếm theo chi phí thống nhất luôn tối ưu vì nó chỉ chọn một đường dẫn có chi phí đường dẫn thấp nhất.

CHƯƠNG 2 – DU LỊCH Ở ROMANIA

1. Greedy Best First Search

2.1.1. Cấu trúc dữ liệu

Theo các nguyên lý của hàm Heuristic thì thuật toán Greedy Best First Search là ưu tiên hàng đâu của việc tìm kiếm ở thời điểm hiện tại. Thuật toán luôn lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải. Đây là thuật toán của sự kết hợp của các thuật toán tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng, đồng thời sử dụng hàm Heuristic để tìm kiếm.

Best First Search cho phép tận dụng lợi thế của cả hai thuật toán. Với sự trợ giúp của Best First Search ở mỗi bước, chúng ta có thể chọn nút hứa hẹn nhất. Trong thuật toán tìm kiếm đầu tiên tốt nhất, chúng tôi mở rộng nút gần nhất với nút mục tiêu và chi phí gần nhất được ước tính bằng hàm heuristic.

Chức năng đánh giá của thuật toán Greedy Best First Search:

***f(n)= h(n).***

h (n) = chi phí để đi từ nút hiện tại nđến nút đích (mục tiêu).

2.1.2. Thuật toán

Thuật toán Greedy Best First Search được thực hiện bởi hàng đợi ưu tiên.

Các bước thuật toán Greedy Best First Search:

Bước 1: Đặt nút bắt đầu vào danh sách MỞ.

Bước 2: Nếu danh sách MỞ trống, Dừng và quay lại thất bại.

Bước 3: Xóa nút n khỏi danh sách MỞ có giá trị thấp nhất là h (n) và đặt nó vào danh sách ĐÃ ĐÓNG.

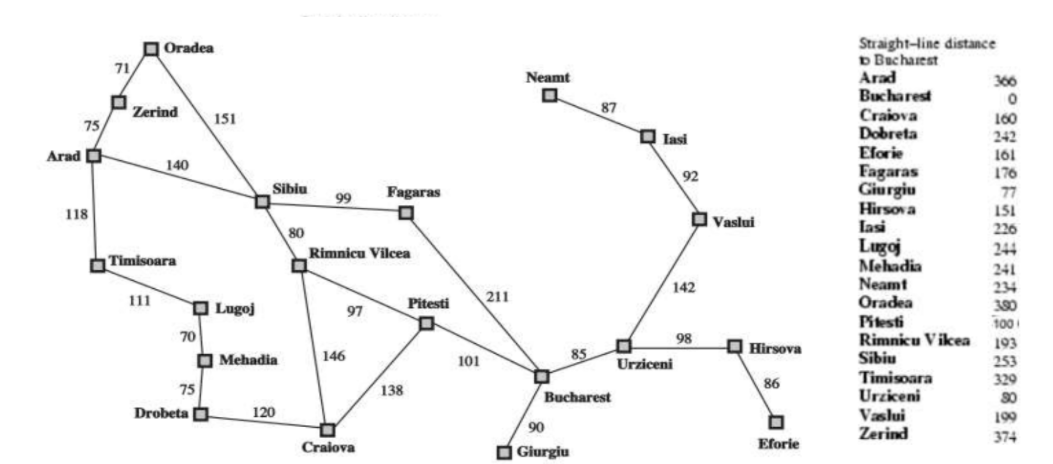
Bước 4: Mở rộng nút n và tạo ra các phần tử kế thừa của nút n.

Bước 5: Kiểm tra từng nút kế thừa của nút n và tìm xem có nút nào là nút mục tiêu hay không. Nếu bất kỳ nút kế thừa nào là nút mục tiêu, sau đó trả về thành công và kết thúc tìm kiếm, nếu không, hãy chuyển sang Bước 6.

Bước 6: Đối với mỗi nút kế thừa, thuật toán kiểm tra hàm đánh giá f (n), sau đó kiểm tra xem nút đã nằm trong danh sách MỞ hoặc ĐÓNG hay chưa.

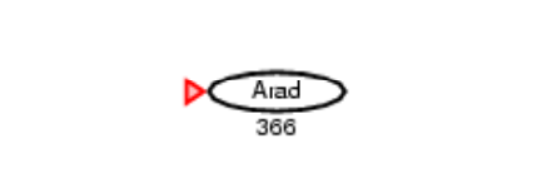
Bước 7: Nếu nút chưa có trong cả hai danh sách, hãy thêm nó vào danh sách MỞ. Quay lại Bước 2.

Trong bài du lịch ở Romania, thì trong bài toán sử dụng h(n) = ước lượng khoảng cách đường thẳng từ thành phố hiện tại n đến Bucharest. Phương pháp sẽ xét các nút có vẻ gần với nút đích nhất vì nó chỉ phát triển từ những nút có ước lượng tốt nhất.



Hình 2. 1: Bản đồ du lịch ở Romania

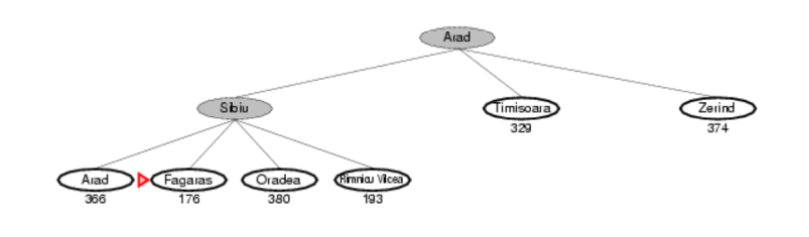
Từ hình trên ta có thể thấy, giả sử ta bắt đầu tử arad, đường đi thực tế đến sibiu = 140. giá sử điểm đích là buncharest, ước lượng từ arad đến đích là 366.



Hình 2. 2: Du lịch ở Romania áp dụng thuật toán GBFS – bước 1

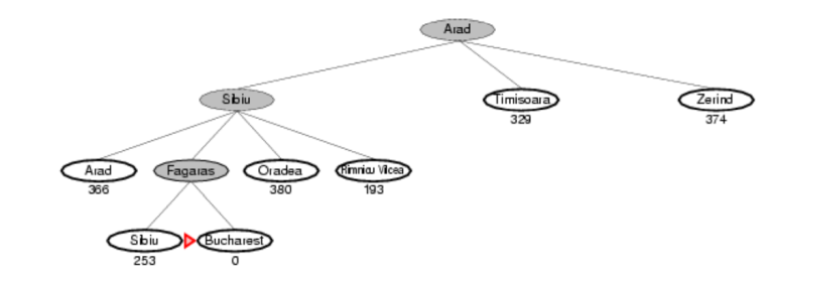
Bước 1: Chọn nút ban đầu, Arad và mở rộng nó

Hình 2. 3: Du lịch ở Romania áp dụng thuật toán GBFS – bước 2

Bước 2: Chọn nút có chi phí thấp nhất - Sibiu và mở rộng nó

Hình 2. 4: Du lịch ở Romania áp dụng thuật toán GBFS – bước 3

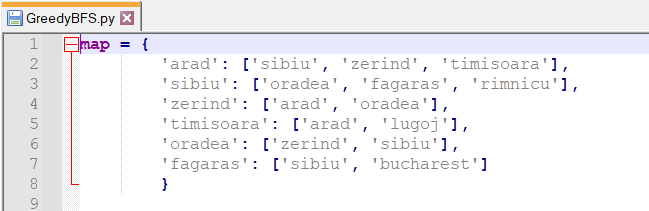
Bước 3: Tương tự như Sibiu thì Fagaras có chi phí nhỏ nhất.



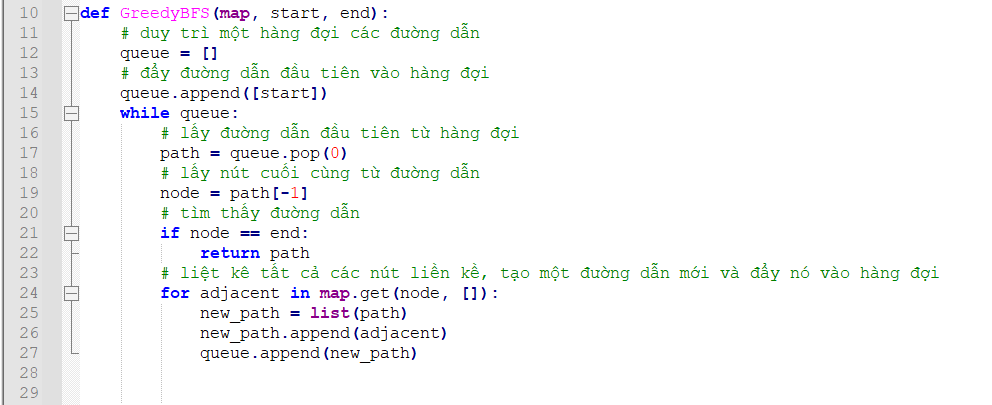
Hình 2. 5: Du lịch ở Romania áp dụng thuật toán GBFS – bước 4

Bước 4: Đến Bucharest. Kết thúc tìm kiếm.

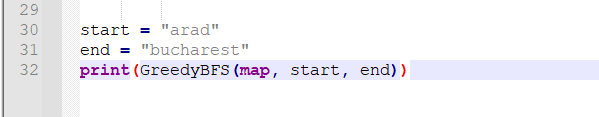
* Thực hiện thuật toán Greedy Best First Search:



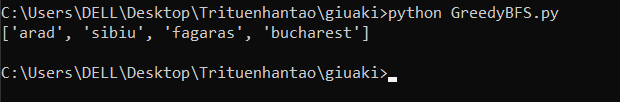
Hình 2. 6: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(1)



Hình 2. 7: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(2)



Hình 2. 8: Code hiện thực của du lịch Romania áp dụng thuật toán Greedy Best First Search(3)



Hình 2. 9: Kết quả của du lịch ở Romania áp dụng thuật toán GBFS

2.1.3. Test

* **Để tối ưu hóa thuật toán Greedy Best First Search thì cần phải thỏa mãn các điều kiện sau:**
* Độ phức tạp về thời gian: Độ phức tạp về thời gian trong trường hợp xấu nhất của tìm kiếm đầu tiên tốt nhất của Greedy là O (bm).
* Độ phức tạp không gian: Độ phức tạp không gian trong trường hợp xấu nhất của tìm kiếm đầu tiên tốt nhất của Greedy là O (bm). Trong đó, m là độ sâu tối đa của không gian tìm kiếm.
* **Đặc điểm của thuật toán Greedy Best First Search:**
* Hoàn thành: Greedy Best First Search cũng không hoàn thành, ngay cả khi không gian trạng thái đã cho là hữu hạn. Vì trong quá trình tìm kiếm có thể vướng (chết tắt) trong các vòng lặp.
* Tối ưu: Nếu phải so sánh thuật toán Greedy Best First Search với các thuật toán tìm kiếm khác thì có thể nói thuật toán Greedy BFS không phải là tối ưu.

1. A\* Algorithm

2.2.1. Cấu trúc dữ liệu

Thuật toán A \* hoạt động dựa trên phương pháp heuristic và điều này giúp đạt được tính tối ưu. A \* là một dạng khác của thuật toán ưu tiên nhất. Tính tối ưu cho phép thuật toán tìm ra giải pháp tốt nhất có thể cho một vấn đề. Các thuật toán như vậy cũng cung cấp tính hoàn chỉnh, nếu có bất kỳ giải pháp nào có thể cho một vấn đề hiện có, thuật toán chắc chắn sẽ tìm thấy nó.

A \* thực sự là một thuật toán rất mạnh được sử dụng để tăng hiệu suất của trí tuệ nhân tạo. Nó là một trong những thuật toán tìm kiếm phổ biến nhất trong AI. Sky là giới hạn khi nói đến tiềm năng của thuật toán này. Tuy nhiên, hiệu quả của thuật toán A \* phụ thuộc nhiều vào chất lượng của hàm heuristic của nó.

Vậy tại sao thuật toán này được ưa thích và sử dụng trong nhiều hệ thống phần mềm? Không có khía cạnh nào của AI mà thuật toán A \* chưa được tìm thấy ứng dụng của nó. Từ tối ưu hóa tìm kiếm đến trò chơi, người máy và máy học, thuật toán A \* là một phần tất yếu của một chương trình thông minh.

Khi A \* đi vào một vấn đề, trước hết nó tính toán chi phí để di chuyển đến các nút lân cận và chọn nút có chi phí thấp nhất. Nếu f (n) biểu thị chi phí, A \* chọn nút có giá trị f (n) thấp nhất. Ở đây ‘n’ biểu thị các nút lân cận. Việc tính toán giá trị có thể được thực hiện như hình dưới đây:

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

*Với:*

* f (n) = chi phí tổng thể ước lượng của đường đi qua nút hiện tại nđến đích
* g (n) = hiển thị giá trị của đường đi ngắn nhất từ ​​nút bắt đầu đến nút n
* h (n) = Sự xấp xỉ heuristic của giá trị của nút

Giá trị heuristic có một vai trò quan trọng trong hiệu quả của thuật toán A \*. Để tìm ra giải pháp tốt nhất, bạn có thể phải sử dụng hàm heuristic khác nhau tùy theo loại vấn đề. Tuy nhiên, việc tạo ra các chức năng này là một nhiệm vụ khó khăn, và đây là vấn đề cơ bản mà chúng ta phải đối mặt trong AI.

2.2.2. Thuật toán A\*

**Các bước của thuật toán A\*:**

Bước 1: Đặt nút bắt đầu trong danh sách MỞ.

Bước 2: Kiểm tra xem danh sách MỞ có trống hay không, nếu danh sách trống thì trả về lỗi và dừng lại.

Bước 3: Chọn nút từ danh sách MỞ có giá trị nhỏ nhất của hàm đánh giá (g + h), nếu nút n là nút mục tiêu thì trả về thành công và dừng lại, ngược lại

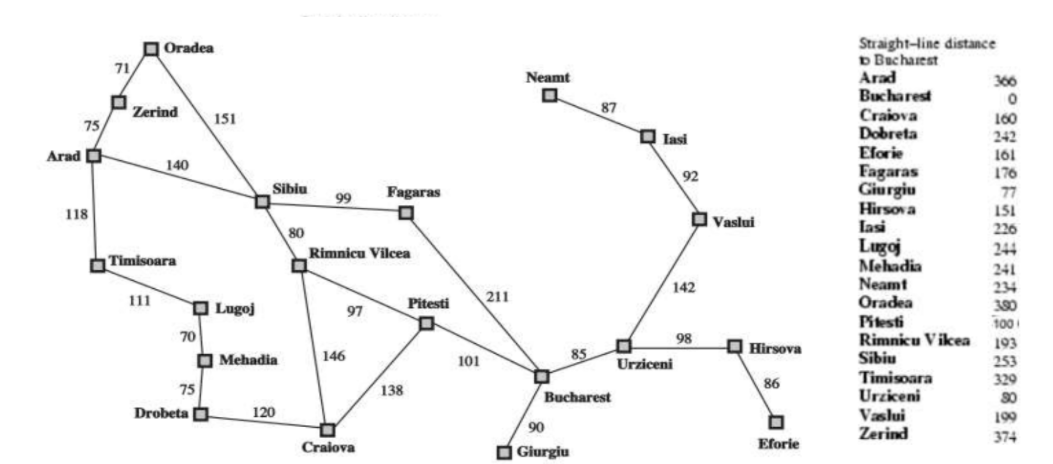
Bước 4: Mở rộng nút n và tạo tất cả các nút kế thừa của nó, và đưa n vào danh sách đóng. Đối với mỗi người kế nhiệm n ', hãy kiểm tra xem n' đã ở trong danh sách MỞ hay ĐÓNG chưa, nếu chưa thì tính toán hàm đánh giá cho n 'và đặt vào danh sách Mở.

Bước 5: Khác nếu nút n 'đã ở trạng thái MỞ và ĐÓNG CỬA,

thì nó sẽ được gắn vào con trỏ quay lại phản ánh giá trị g (n ') thấp nhất.

Bước 6: Quay lại Bước 2.

Giả sử, trong du lịch ở Romania có đồ thị chi phí đường đi và ước lượng tới Bucharest:



Hình 2. 10: Bản đồ du lịch ở Romania

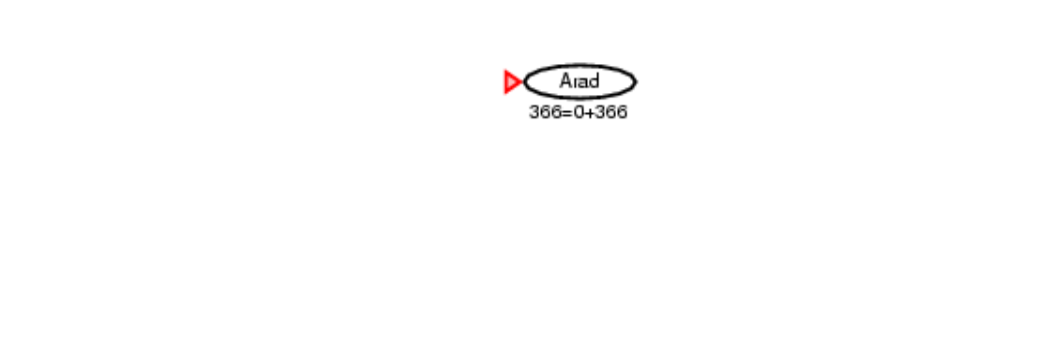
Khi đó, các thực hiện của A\* khi tìm đường từ arad đến Buncharest như sau:

Bước 1: Tính tổng chi phí ước tính của nút ban đầu, Arad

h (Arad) = 366

g (Arad) = 0

f (Arad) = 366 + 0 = 366

Mở rộng Arad

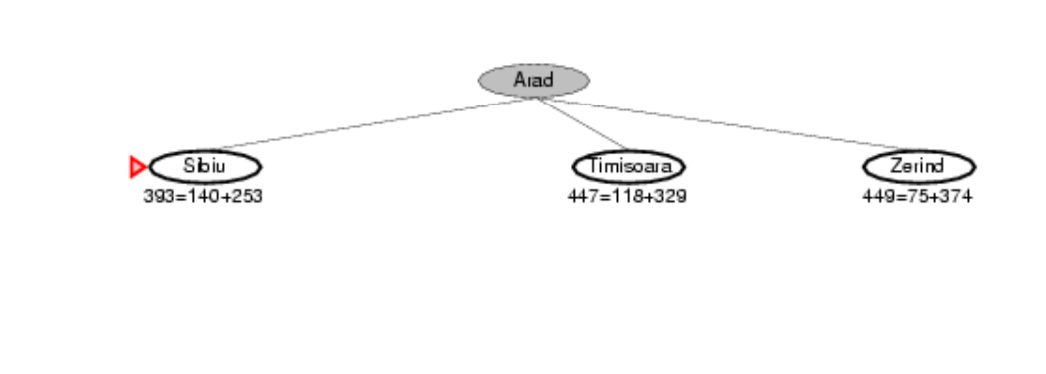
Hình 2. 11: Du lịch ở Romania áp dụng thuật toán A\* - bước 1

Bước 2: Tính toán tổng chi phí ước tính của trạng thái được tạo ra:

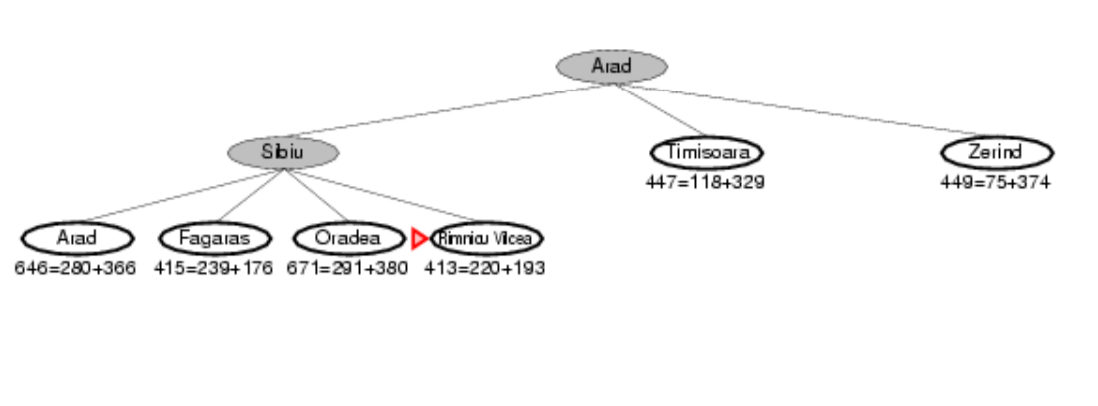
f (Sibiu) = g (Sibiu) + h (Sibiu) = 140 + 253 = 393

f (Timisoara) = 118 + 329 = 447

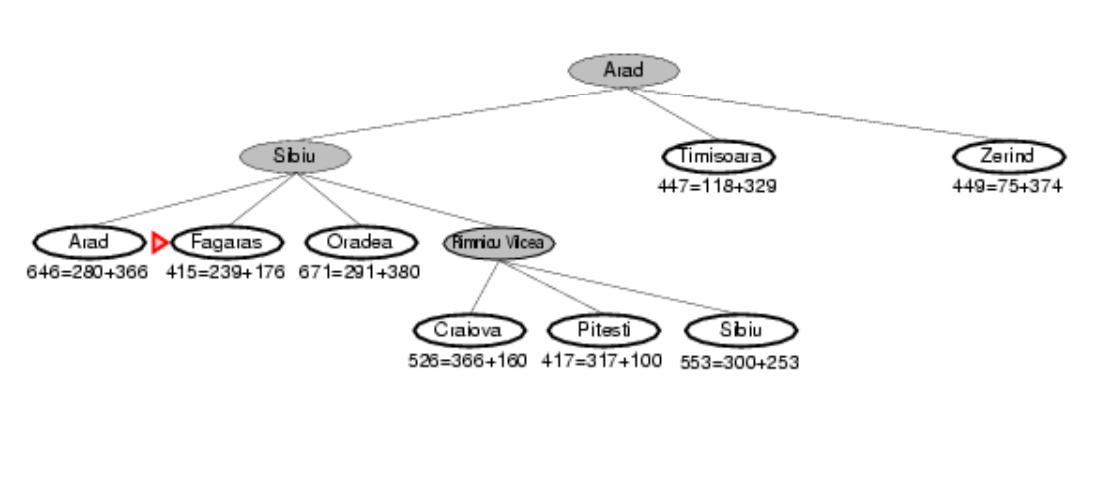
f (Zerind) = 75 + 374 = 449

Mở rộng Sibiu.

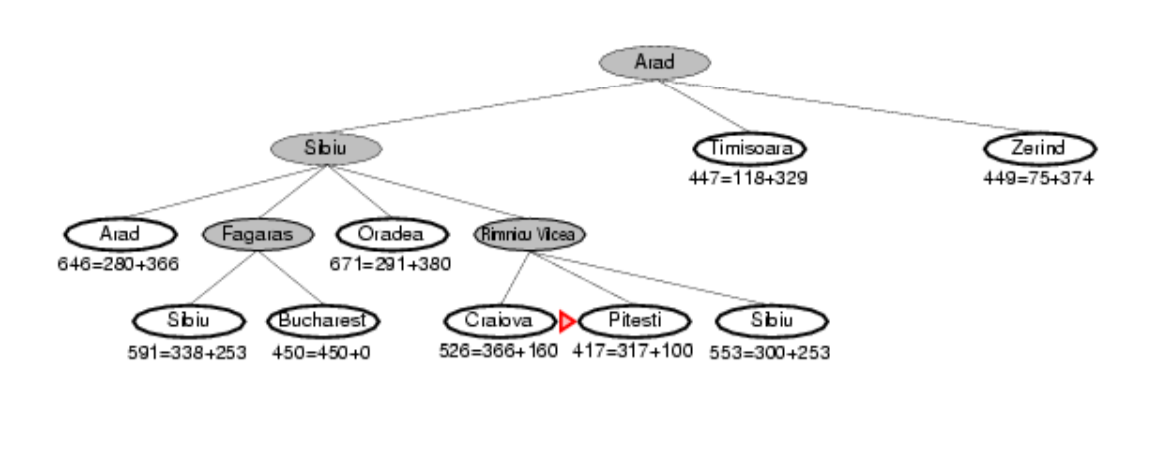
Hình 2. 12: Du lịch ở Romania áp dụng thuật toán A\* - bước 2

Bước 3: Mở rộng Rimnicu Vilicea và xác định f (n) cho mỗi nút n

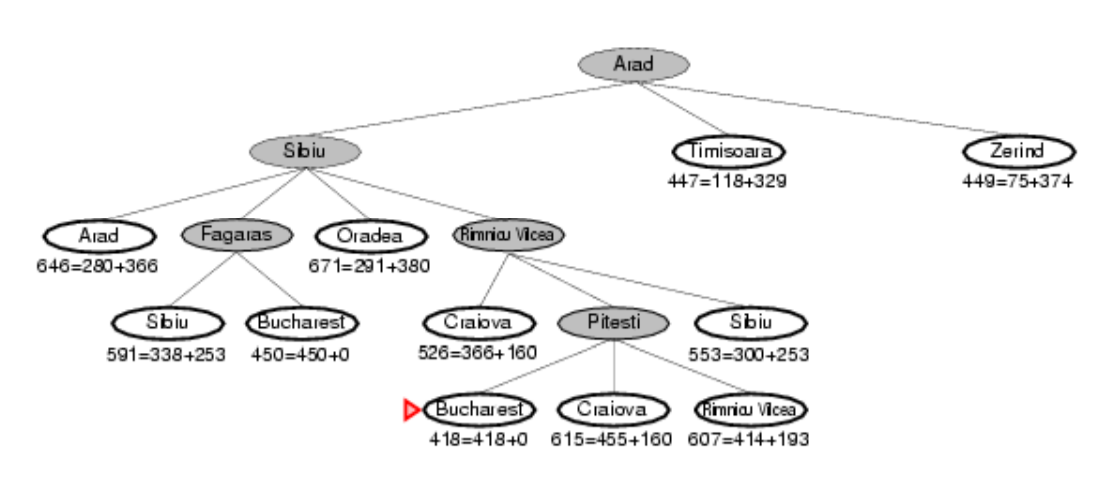
Hình 2. 13: Du lịch ở Romania áp dụng thuật toán A\* - bước 3

Bước 4: Mở rộng Fagaras và xác định f (n) cho mỗi nút n

Hình 2. 14: Du lịch ở Romania áp dụng thuật toán A\* - bước 4

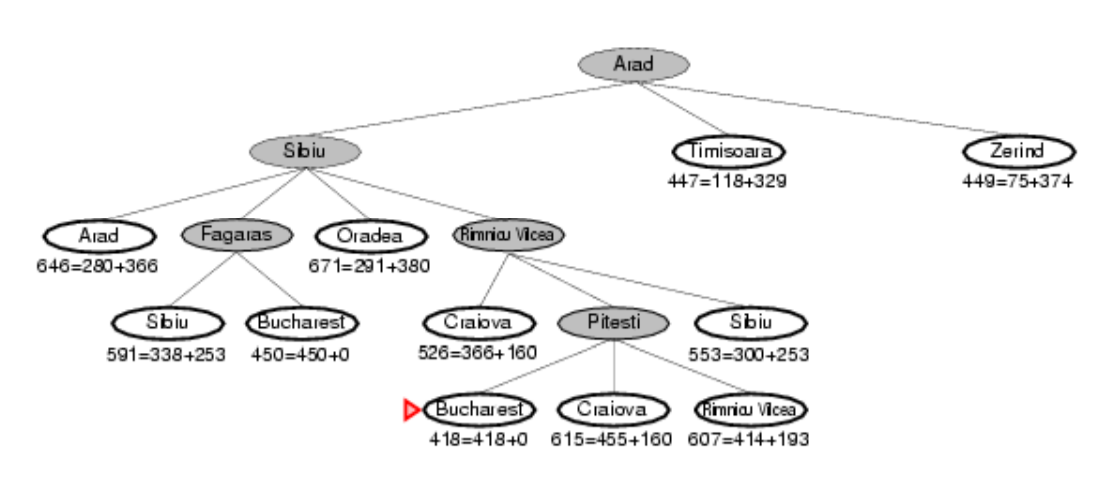
Bước 5: Mở rộng Pitesti và xác định f (n) cho mỗi nút n

Hình 2. 15: Du lịch ở Romania áp dụng thuật toán A\* - bước 5

Bước 6: Mở rộng Pitesti và xác định f (n) cho mỗi nút n

Hình 2. 16: Du lịch ở Romania áp dụng thuật toán A\* - bước 6

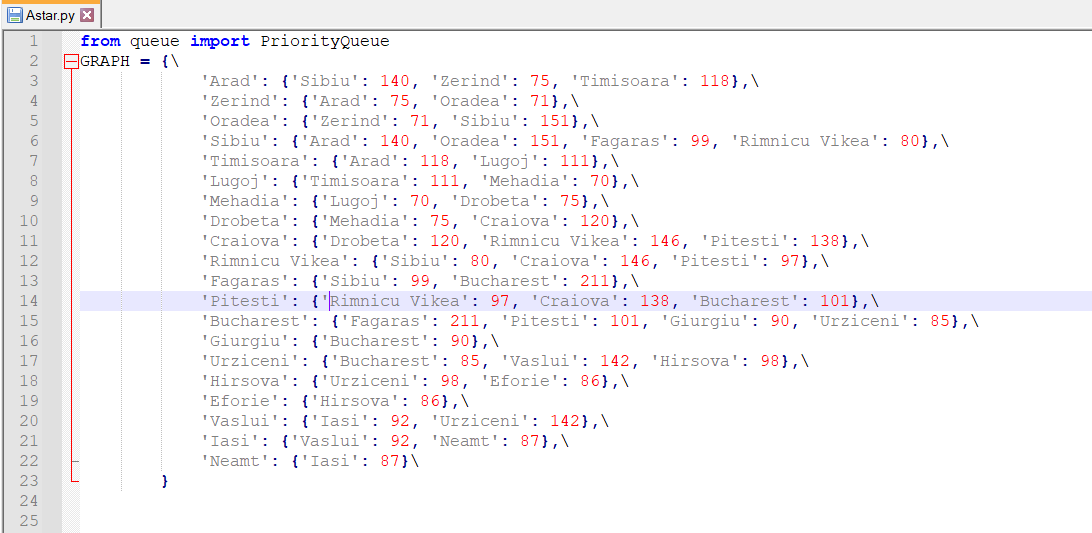
Bước 7: So sánh chi phi của các thành phố và xác định được Bucharest có chi phí thấp nhất



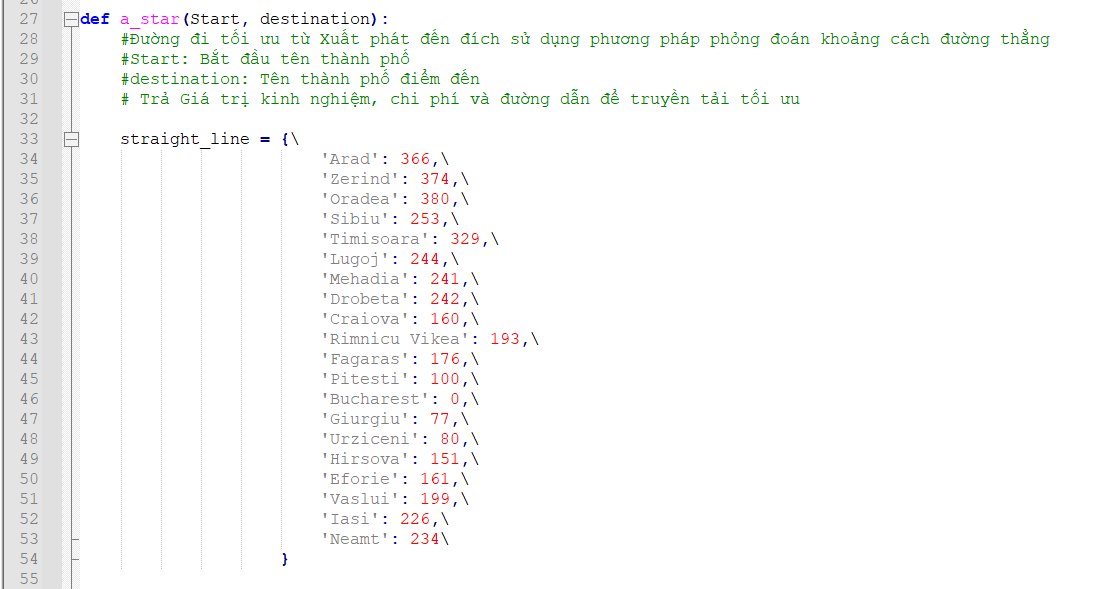
Hình 2. 17: Du lịch ở Romania áp dụng thuật toán A\* - bước 7

Sau khi đến đích, ta sẽ tiến hành truy vấn ngược lại để tìm đường đi tối ưu từ Arad đên Bucharest.

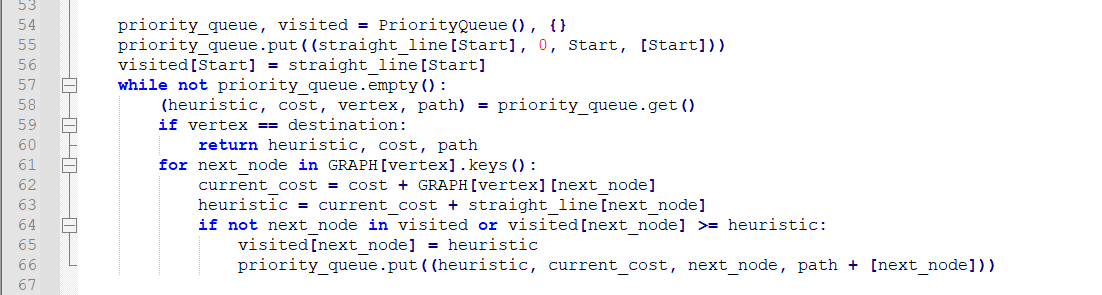
* Thực hiện thuật toán A\*:



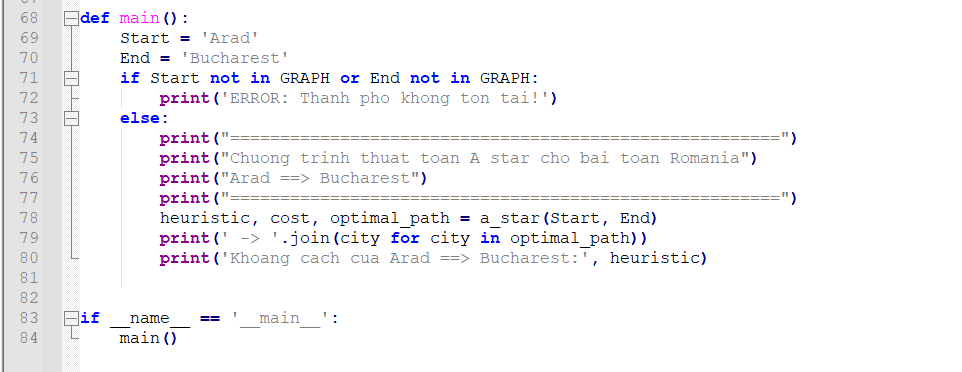
Hình 2. 18: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(1)



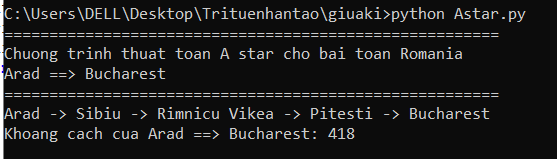
Hình 2. 19: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(2)



Hình 2. 20: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(3)



Hình 2. 21: Code hiện thực du lịch ở Romania áp dụng thuật toán A\*(4)



Hình 2. 22: Kết quả du lịch ở Romania áp dụng thuật toán A\*

2.2.3. Test

A \* là tên được đặt cho thuật toán mà hàm h (n) có thể chấp nhận được. Nói cách khác, nó được đảm bảo cung cấp mức đánh giá thấp hơn chi phí thực sự cho mục tiêu. A \* là tối ưu và đầy đủ. Nói cách khác, nó được đảm bảo để tìm ra một giải pháp, và giải pháp đó được đảm bảo là giải pháp tốt nhất.

Trên thực tế, A \* chỉ hoàn chỉnh nếu cây mà nó đang tìm kiếm có hệ số phân nhánh hữu hạn và không chứa đường dẫn có chi phí hữu hạn, có vô số nút dọc theo nó. Cả hai điều kiện này đều có khả năng được đáp ứng trong tất cả các tình huống trong thế giới thực, và vì vậy để đơn giản hơn có thể nói rằng A \* là hoàn chỉnh nếu như A \* hoàn tất nếu biểu đồ mà nó đang tìm kiếm là hữu hạn cục bộ (nghĩa là nó có hệ số phân nhánh hữu hạn) và nếu mọi cung giữa hai nút trong biểu đồ có chi phí khác không.

* **A \* là tối ưu nếu có thể được chứng minh bằng cách xem xét một ví dụ ngược lại:**

Hãy tưởng tượng chúng ta đang áp dụng thuật toán A \* cho một biểu đồ có hai mục tiêu, G1 và G2. Chi phí đường đi của G1 là f1 và chi phí đường đi của G2 là f2, trong đó f2> f1. G1 là mục tiêu với chi phí thấp hơn, nhưng chúng ta hãy tưởng tượng một kịch bản trong đó thuật toán A \* đã đạt đến G2 mà không cần khám phá G1. Nói cách khác, chúng tôi đang hình dung một tình huống trong đó thuật toán không chọn mục tiêu với chi phí thấp hơn.

Nếu chúng ta coi một nút, n, nằm trên đường đi tối ưu từ nút gốc đến G1, thì vì h là một heuristic có thể chấp nhận được:

***f1 ≥ f (n)***

Lý do duy nhất mà thuật toán không chọn mở rộng n trước khi nó đạt đến G2 là nếu

***f (n) > f (G2)***

Do đó, bằng cách kết hợp hai biểu thức này với nhau, chúng ta sẽ được

***f1 ≥ f(G2)***

Bởi vì G2 là một trạng thái mục tiêu, nó phải là trường hợp h (G2) = 0, và do đó f (G2) = g (G2). Vì vậy, chúng tôi có

***f1 ≥ g(G2)***

Do đó, điều này mâu thuẫn với giả định ban đầu của chúng tôi rằng G2 có chi phí đường đi cao hơn G1, điều này chứng tỏ rằng A \* chỉ có thể chọn con đường có chi phí thấp nhất để đạt được mục tiêu.

A\* đã từng được đề cặp tương tự như Breadth First Search (tìm kiếm đầu tiên theo chiều rộng). Trên thực tế, tìm kiếm đầu tiên theo chiều rộng có thể được coi là một trường hợp đặc biệt của A \*, trong đó h (n) luôn bằng 0, do đó f (n) = g (n), và trong đó mọi đường dẫn trực tiếp giữa nút và người kế nhiệm trực tiếp của nó có chi phí là 1.

Tóm lại, A \* là hoàn chỉnh, tối ưu và nó có độ phức tạp về thời gian và không gian là O (bm) không như Greedy Best First Search không hoàn chỉnh, không tối ưu, có độ phức tạp thời gian là O (bm) và độ phức tạp không gian có thể là đa thức.

TỰ ĐÁNH GIÁ

|  |  |  |
| --- | --- | --- |
| Bài tập | Nội dung | Điểm |
| **Bài 1: Tìm số hòn đảo**  **(5 điểm)** | **Giải thuật Breadth First Search**  **(2.5 đ)** |  |
| Cấu trúc dữ liệu (0.75 đ) | 0.5 |
| Thuật toán (1) | 1 |
| Test (0.75 đ) | 0.75 |
| **Giải thuật Uniform Cost Search**  **(2.5 đ)** |  |
| Cấu trúc dữ liệu (0.75 đ) | 0.5 |
| Thuật toán (1) | 0.25 |
| Test (0.75 đ) | 0.75 |
| **Bài 2: Du lịch ở Romania**  **(5 điểm)** | **Giải thuật Greedy Best First Search (2.5 đ)** | 2.5 |
| Cấu trúc dữ liệu (0.75 đ) | 0.75 |
| Thuật toán (1) | 0.75 |
| Test (0.75 đ) | 0.5 |
| **Giải thuật A\* (2.5 đ)** |  |
| Cấu trúc dữ liệu (0.75 đ) | 0.5 |
| Thuật toán (1) | 1 |
| Test (0.75 đ) | 0.75 |
| **Tổng điểm** | 10 đ | 8.25 |

Bảng 2. 1: Tự đánh giá quá trình làm bài

TÀI LIỆU THAM KHẢO

Chương 1:

1. <https://www.geeksforgeeks.org/islands-in-a-graph-using-bfs/>
2. <https://vnoi.info/wiki/algo/graph-theory/breadth-first-search.md>
3. <http://futuresoft.yolasite.com/resources/Artificial%20Intelligence%20Illuminated.pdf>
4. <https://labs.flinters.vn/algorithm/algorithm-cac-thuat-toan-tim-kiem-trong-ai/>
5. <https://huuvinhfit.files.wordpress.com/2015/01/chuong-4-tim-kiem-heuristic.pdf>
6. <https://www.mygreatlearning.com/blog/best-first-search-bfs/#bestfirstsearchalgorithm>
7. https://vi.wikipedia.org/wiki/Tìm\_kiếm\_theo\_lựa\_chọn\_tốt\_nhất
8. <https://www.javatpoint.com/ai-uninformed-search-algorithms>
9. <https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/?ref=lbp>
10. <https://www.educba.com/uniform-cost-search/>
11. <https://stackoverflow.com/questions/44151713/what-is-the-difference-between-uniform-cost-search-and-best-first-search-methods>
12. <https://www.geeksforgeeks.org/search-algorithms-in-ai/?ref=lbp>
13. <https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/>

Chương 2:

1. <http://futuresoft.yolasite.com/resources/Artificial%20Intelligence%20Illuminated.pdf>
2. <https://www.youtube.com/watch?v=Y4tPnavVwKM>
3. <https://huuvinhfit.files.wordpress.com/2015/01/chuong-4-tim-kiem-heuristic.pdf>
4. <https://www.geeksforgeeks.org/a-search-algorithm/?ref=leftbar-rightbar>
5. <https://drive.google.com/file/d/1iR1-CP6JmRSJTPMFSPDCqHQ7X6QlTaWf/view>
6. <https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/>
7. <http://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf>
8. <http://web.pdx.edu/~arhodes/ai6.pdf>
9. <https://www.javatpoint.com/ai-informed-search-algorithms>
10. https://vi.wikipedia.org/wiki/Giải\_thuật\_tìm\_kiếm\_A\*
11. <https://bloglaptrinh2016.wordpress.com/2016/06/09/thuat-toan-tim-kiem-greedy-best-first-search/>
12. <https://notebook.community/jo-tez/aima-python/search>
13. <https://www.geeksforgeeks.org/search-algorithms-in-ai/?ref=lbp>