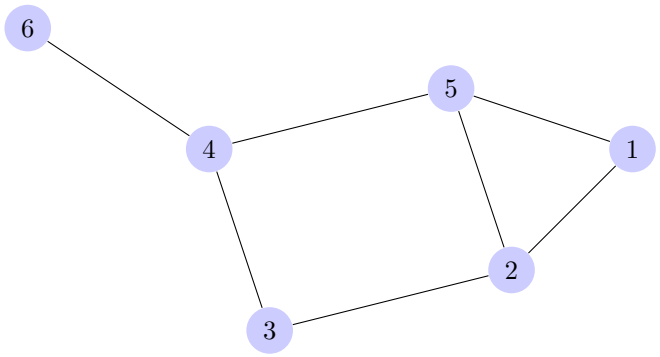
ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN KHOA HỌC MÁY TÍNH

FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CƠ SỞ TRÍ TUỆ NHÂN TẠO

PROJECT REPORT 01 - BÁO CÁO ĐỒ ÁN 01

SEARCH ON GRAPH - TÌM KIẾM TRÊN ĐỒ THỊ



Giảng viên lý thuyết
GS.TS Lê Hoài Bắc

Giảng viên hướng dẫn
Dương Nguyễn Thái Bảo, Nguyễn Ngọc Đức, Hoàng Xuân Trường

Người thực hiện

Họ và tên: Lê Nhựt Nam
Mã số sinh viên: 18120061
Email: 18120061@student.hcmus.edu.vn

Tháng 10 năm 2020

Lời cảm ơn

Trong quá trình thực hiện đồ án này, em đã nhận được rất nhiều sự giúp đỡ cũng như hỗ trợ từ các thầy cô Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM và các bạn bè trong lớp Cơ sở Trí tuệ Nhân tạo. Em xin bày tỏ lòng cảm ơn chân thành đến mọi người vì đã giúp đỡ hướng dẫn, chỉ bảo rất tận tình.

Đặc biệt, em xin bày tỏ lòng biết ơn sâu sắc đến các thầy cô khoa Công nghệ Thông tin, cụ thể hơn là thầy Lê Hoài Bắc và các thầy hướng dẫn đã giảng dạy rất kĩ lưỡng, cung cấp nhiều slides, tài nguyên học tập cần thiết, tạo điều kiện tốt nhất để bản thân em có thể hoàn thành được đồ án này.

Trong quá trình, viết báo cáo này, bản thân em không thể tránh khỏi nhiều sai sót về mặt chuyên môn phân tích thuật toán, nguồn thông tin có thể sai sót và nhất là việc trình bày code có thể chưa hoàn thiện về mặt "clean code", ... Em mong nhận được góp ý để hoàn thiện hơn đối với đồ án này, cũng như rút kinh nghiệm cho những đồ án kế tiếp.

Đại học Khoa học Tự nhiên, ĐHQG-HCM.
Lê Nhựt Nam
Tháng 10 năm 2020,

Mục lục

1 Tổng quan về đồ án

1.1 Mục tiêu của đồ án

Nghiên cứu, cài đặt và trình bày các thuật toán tìm kiếm trên đồ thị

1.2 Các yêu cầu

- Thông tin sinh viên: họ tên, MSSV...
- Mức độ hoàn thành của mỗi mức yêu cầu. Tự đánh giá đồ án trên thang điểm 10.
- Trình bày lý thuyết cơ bản (ý tưởng, độ phức tạp, tính chất,...) của mỗi thuật toán cài đặt (BFS, DFS, UCS, A^*).
- Trình bày điểm khác biệt giữa UCS và A*.
- Thuật toán tìm kiếm khác ngoài 4 thuật toán (BFS, DFS, UCS, A^*): Greedy Best-First Search

1.3 Bảng tự đánh giá mức độ hoàn thành công việc

STT	Tiêu chí	Mức độ	Điểm	Ghi chú
1	Trình bày thuật toán BFS	100%	10	
2	Trình bày thuật toán DFS	100%	10	
3	Trình bày thuật toán UCS	100%	10	
4	Trình bày thuật giải A*	100%	10	
5	Trình bày các sinh trọng số để chạy UCS và A*	100%	10	
6	Trình thuật toán Greedy Best-First Search	100%	10	Điểm cộng
7	Trình bày điểm khác biệt giữa UCS và A*	100%	10	
8	Chạy thử và có video demo (Ít nhất 3 testcase) cho mỗi thuật toán	100%	10	
9	Có trình bày báo cáo	100%	10	

2 Lý thuyết cơ bản

2.1 Thuật toán tìm kiếm theo chiều rộng - Breadth First Search

Thuật toán Breadth First Search (BFS - Tìm kiếm theo chiều rộng) là thuật toán xét (duyệt) hoặc tìm kiếm trên cây và đồ thị, có chiến lược tìm kiếm mù (tìm kiếm không có định hướng, không chú ý đến thông tin, giá trị được duyệt).

2.1.1 Ý tưởng

Thuật toán có chiến lược tìm kiếm đơn giản bằng việc xây dựng cây tìm kiếm theo chiều rộng, nút gốc được mở rộng đầu tiên, sau đó tất cả những nút con của nút gốc sẽ được mở rộng, tiếp theo và tiếp sau các nút con của những nút con đó sẽ được mở rộng.
Tổng quát: Tất cả các nút ở độ sâu d trong cây tìm kiếm sẽ được mở rộng trước những nút ở độ sâu d + 1

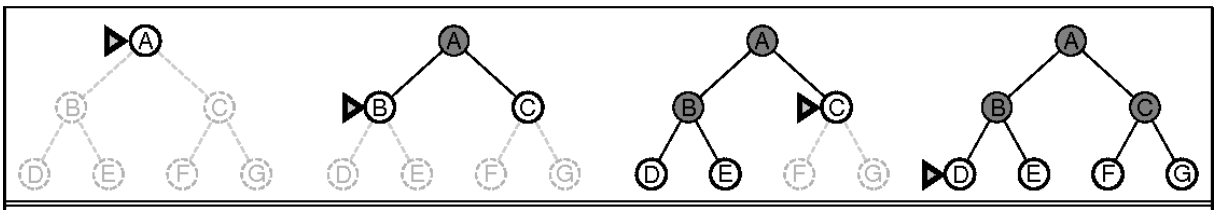


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

2.1.2 Mã giả - Pseudocode

Algorithm 1 Thuật toán tìm kiếm theo chiều rộng - Breadth-First Search Algorithm (Artificial Intelligence: A Modern Approach (3rd Edition) - Breadth-First Search Pseudocode) - P82

Input: Bài toán/ Vấn đề, Ngăn chứa (FIFO queue)
Output: Lời giải cho bài toán hoặc không có lời giải (Thất bại)
Data: Testing set x
function BREADTH-FIRST-SEARCH(problem) **return** *a solution, or failure*
node ← a node with State = problem.Initial-State, Path-Cost = 0
if *problem.Goal-Test(node.State)* **then**
  return *Solution(node)*
end
frontier ← node
explored ← ∅
loop do
if *frontier* = ∅ **then**
  return *Failure*
end
node ← Pop(frontier)
add node.State to explored
foreach *action* **in** *problem.Actions(node.State)* **do**
  child ← Child-Node(problem, node, action)
  if *child.State* **is not in** *explored* **or** *frontier* **then**
    if *problem.Goal-Test(child.State)* **then**
      return *Solution(child)*
    end
    frontier ← Insert(child, frontier)
  end
end

2.1.3 Đánh giá thuật toán

Đầu vào tìm kiếm theo chiều rộng được giả định là một đồ thị hữu hạn, được biểu diễn một cách rõ ràng dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự .Nhưng trong lý thuyết, ta phải ngầm định rằng đầu vào có thể là một biểu diễn ngầm định của một đồ thị vô hạn.

2.1.3.1 Tính đầy đủ Breadth-First Search có tính đã đầy đủ - nếu nút mục tiêu nông nhất ở độ sâu d, nó cuối cùng sẽ tìm thấy nó sau khi mở rộng tất cả các nút nông hơn d.

2.1.3.2 Tính tối ưu Breadth-First Search dừng lại ở mục tiêu nông nhất được tìm thấy, mục tiêu nông nhất này có thể không nhất thiết là tối ưu nhất. Nên trường hợp tổng quát, nó sẽ không tối ưu

2.1.3.3 Độ phức tạp về thời gian Với b là số nút con của những nút không phải đích đến, d là độ sâu khi tìm thấy đích.
Ta có độ phức tạp thuật toán: $1 + b^1 + b^2 + b^3 + ... + b^d = O(b^d)$

2.1.3.4 Độ phức tạp về không gian Với b là số nút con của những nút không phải đích đến, d là độ sâu khi tìm thấy đích.
Tất cả các trạng thái phải được ghi nhớ để tránh các đường dẫn dư thừa và lặp lại. Do đó, cho đến khi đạt đến nút mục tiêu ở độ sâu d, tất cả các nút cho đế n$d-1$ phải được lưu trong bộ nhớ. Ta có độ phức tạp về không gian: $b^1 + b^2 + b^3 + ... + b^d = O(b^d)$ ($O(b^{d-1})$ cho những đỉnh đã được duyệt lưu trong explored và $O(b^d)$ cho hàng đợi)

2.2 Thuật toán tìm kiếm theo chiều sâu - Depth First Search

2.2.1 Ý tưởng

Thuật toán bắt đầu ở nút gốc (chọn một số nút tùy ý làm nút gốc trong trường hợp biểu đồ) và khám phá càng xa càng tốt dọc theo mỗi nhánh trước khi backtracking.

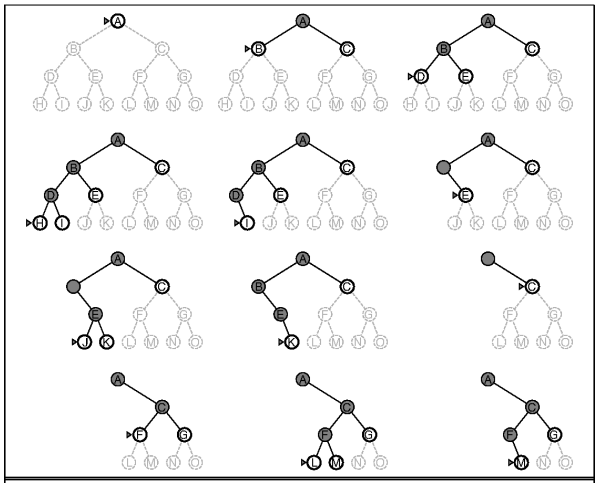


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors, and M is the only goal node.

Hình 1: Artificial Intelligence: A Modern Approach (3rd Edition) Depth First Search

2.2.2 Đánh giá thuật toán

Đầu vào DFS được giả định là một đồ thị hữu hạn, được biểu diễn một cách rõ ràng dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự .Nhưng trong lý thuyết, ta phải ngầm định rằng đầu vào có thể là một biểu diễn ngầm định của một đồ thị vô hạn.

2.2.2.1 Tính đầy đủ: Thuật toán DFS không đảm bảo tính đầy đủ nếu không gian tìm kiếm là vô hạn, khi không gian tìm kiếm hữu hạn thì ngược lại.

2.2.2.2 Tính tối ưu: Thuật toán DFS không đảm bảo tính tối ưu

2.2.2.3 Độ phức tạp về thời gian Tăng theo hàm mũ $O(b^m)$, xấu là khi m lớn hơn nhiều lần so với độ sâu d, tốt nhất khi lời giải không ở quá sâu

2.2.2.4 Độ phức tạp về không gian Gần như là tuyến tính $O(bm)$

2.2.3 Mã giả - Pseudocode thuật toán DFS có giới hạn

Dựa trên cơ sở của DFS, nhưng có một giới hạn về độ sâu để tránh khỏi rơi vào vòng lặp vô hạn, các nút con của các nút ở độ sâu L có thể coi là không có con.

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Figure 3.17 A recursive implementation of depth-limited tree search.

Hình 2: Artificial Intelligence: A Modern Approach (3rd Edition) Depth Limit Search Pseudocode

Đánh giá:

- Tính đầy đủ: Nếu L $<$ d thì thuật toán có tính đầy đủ
- Tính tối ưu: Nếu L $<$ d thì thuật toán có tính tối ưu và không có khi L $>$ d
- Độ phức tạp về thời gian: $O(b^L)$
- Độ phức tạp về không gian: $O(bL)$

2.3 Thuật toán tìm kiếm đồng nhất - Uniform Cost Search

2.3.1 Hàng đợi ưu tiên - Priority Queue

Hàng đợi ưu tiên là một cấu trúc dữ liệu trong đó ta có thể thêm và lấy ra các phần tử dựa trên độ ưu tiên của mỗi phần tử.
Trong bài lab này, sử dụng module queue với class PriorityQueue của Python, các thao tác thêm và lấy đều có độ phức tạp là $O(logN)$ với N là số phần tử trong hàng đợi ưu tiên.

2.3.2 Hàm chi phí - Cost Function

Ta có hàm tính chi phí:
f(n) = tổng chi phí đường đi từ nút start đến nút n
Đối với lab, khoảng cách Euclidean cho ta khoảng cách ngắn nhất trong các loại khoảng cách giữa hai điểm trong toán học nên sẽ dùng khoảng cách này để tính trọng số giữa 2 đỉnh trong đồ thị

2.3.3 Ý tưởng

2.3.4 Mã giả - Pseudocode

Algorithm 2 Thuật toán chi phí đồng nhất - Uniform-cost Search Algorithm (Artificial Intelligence: A Modern Approach (3rd Edition) - Uniform-cost Search Pseudocode) - P84

Input: Bài toán/ Vấn đề, Ngăn chứa (FIFO queue)
Output: Lời giải cho bài toán hoặc không có lời giải (Thất bại)
Frontier is a priority queue

```
function UNIFORM-COST-SEARCH((problem) return a solution, or failure
node ← a node with State = problem.Initial-State, Path-Cost = 0
frontier ← node /* a priority queue ordered by Path-Cost */
explored ← ∅
 loop do
 if frontier = ∅ then
 |  return Failure
end
node ← Pop(frontier) /* chooses the lowest-cost node in frontier */
if problem.Goal-Test(node.State) then
 |  return Solution(node)
end
add node.State to explored
 foreach action in problem.Actions(node.State) do
 |  child ← Child-Node(problem, node, action)
 |  if child.State is not in explored or frontier then
 |  |  frontier ← Insert(child, frontier)
 |  end
 |  else if child.State in explored with higher Path-Cost then
 |  |  replace that frontier node with child
 |  end
end
```

2.3.5 Đánh giá thuật toán

Đầu vào tìm kiếm tìm kiếm đồng nhất được giả định là một đồ thị hữu hạn, được biểu diễn một cách rõ ràng dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự .Nhưng trong lý thuyết, ta phải ngầm định rằng đầu vào có thể là một biểu diễn ngầm định của một đồ thị vô hạn.

Đánh giá chung:

- Thuộc loại tìm kiếm mù (không có thông tin) và có thể đưa ra giải pháp tối ưu cho bài toán và khá tương tự với Heuristic Search khi mà hàm h(n) = 0 (một hàm hằng, bằng 0).
- Với UCS, chắc chắn sẽ tìm thấy lời giải (đường đi) nếu bài toán tồn tại lời giải và đường đi (lời giải) đó là đường đi (lời giải) tốt nhất.
- Tuy nhiên, điểm yếu của thuật toán này là tốn nhiều không gian lưu trữ và nó có thể bị mắc kẹt trong một vòng lặp vô hạn vì nó xem xét mọi con đường có thể đi từ nút gốc (trạng thái bắt đầu) đến nút đích(trạng thái kết thúc).

2.3.5.1 Tính đầy đủ Nếu chi phí giữa mỗi đỉnh là một hằng số dương thì tính đầy đủ của thuật toán tìm kiếm chi phí đồng nhất sẽ được đảm bảo.

2.3.5.2 Tính tối ưu Thuật toán đảm bảo được tính tối ưu, vì ở mỗi bước, đường dẫn với chi phí thấp nhất được chọn và đường dẫn không bao giờ ngắn hơn khi các nút được thêm vào, đảm bảo rằng tìm kiếm mở rộng các nút theo thứ tự chi phí đường dẫn tối ưu của chúng.

2.3.5.3 Độ phức tạp về thời gian Với b là số nút con của những nút không phải đích đến. Với C^* là là chi phí đường đi tối ưu của lời giải và mỗi bước tiêu tốn chi phí ít nhất là e. Thuật toán có độ phức tạp về thời gian là:

$$O(b^{1+[\frac{C*}{e}]})$$

2.3.5.4 Độ phức tạp về không gian Với b là số nút con của những nút không phải đích đến. Với C^* là là chi phí đường đi tối ưu của lời giải và mỗi bước tiêu tốn chi phí ít nhất là e. Thuật toán có độ phức tạp về không gian là:

$$O(b^{1+[\frac{C*}{e}]})$$

2.4 Tìm kiếm A^* - A^* Search

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.
Hàm Heuristic là hàm ứng với mỗi trạng thái hay mỗi sự lựa chọn một giá trị ý nghĩa đối với vấn đề dựa vào giá trị hàm này ta lựa chọn hành động.
Tìm kiếm A^* (Đọc là: a star) là mộ hình thức của tìm kiếm tối ưu (Best-First Search) được biết đến nhiều nhất
A^* là một thuật giải đánh giá một nút dựa trên chi phí từ nút gốc đến nút đó - $g(n)$ cộng với một ước lượng chi phí từ nút đó đến đích - $h(n)$
Evaluation function:

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

Trong đó:
- $g(n)$ là chi phí từ nút gốc đến nút n hiện tại.
- $h(n)$ là ước lượng chi phí ngắn nhất để đi từ nút hiện tại n đến đích.
- $f(n)$ là ước lượng chi phí của lời giải "tốt nhất" qua n.

Trường hợp đặc biệt của A^*
Trường hợp hàm đánh giá (Evaluation function) của chúng ta là:

$$f(n) = g(n)$$

Đó là Uniform-Cost Search (Tìm kiếm với chi phí đồng nhất với các thông tin đỉnh, cung, giá thành cung, thực hiện trên cây. $\Longrightarrow A^T$
Trường hợp hàm đánh giá (Evaluation function) của chúng ta là:

$$f(n) = h(n)$$

Đó là tìm kiếm tham lam Heuristic với các thông tin đỉnh, cung, giá thành cung, tri thức bổ sung từ hàm $h(n)$, thực hiện trên cây. $\Longrightarrow A^{KT}$

2.4.1 Ý tưởng

Thuật giải A* cân bằng lại trọng số cho đồ thị và sâu đó tìm kiếm bằng thuật toán Dijsktra (có thể thay bằng những thuật toán khác) trên đồ thị mới này, giảm thiểu số nút phải mở ra
Chọn hàm Heuristic đối với lab:
Python code cho một số công thức khoảng cách toán học:

```
def euclidean_distance(current_x, current_y, goal_x, goal_y):
    """
    Euclidean distance
    current_node(current_x, current_y)
    goal_node(goal_x, goal_y)
    """
    distance = math.sqrt((goal_x - current_x)**2 + (goal_y - current_y)**2)
    return distance
```

Khoảng cách Euclidean cho ta khoảng cách ngắn nhất trong các loại khoảng cách giữa hai điểm trong toán học do nó chính là chiều dài của đoạn thẳng nối hai điểm, hướng mở rộng đường đi của vật thể cũng không bị giới hạn. Đây cũng là Heuristic đơn giản nhất. Nhưng nó gây phức tạp hơn khi phải tính toán bình phương và lấy căn bậc hai, trong khi đồ thị của chúng ta biểu diễn trên khung hình PyGame với tọa độ nguyên.

```
def diagonal_distance(current_x, current_y, goal_x, goal_y):
    """
    Diagonal distance
    current_node(current_x, current_y)
    goal_node(goal_x, goal_y)
    """
    return max(abs(current_x - goal_x),  abs(current_y - goal_y))
```

```
def manhattan_distance(current_x, current_y, goal_x, goal_y):
    """
    Manhattan distance
    current_node(current_x, current_y)
    goal_node(goal_x, goal_y)
    """
    return abs(current_x - goal_x) + abs(current_y - goal_y)
```

Khoảng cách Manhattan và Diagonal đơn giản hơn, vì lược giản được bình phương và căn bậc hai, trong đó Manhattan cho ta hàm heuristic có giá trị nhỏ hơn. Nên trong bài lab sử dụng khoảng cách Manhattan

2.4.2 Mã giả - Pseudocode

Thuật giải A*:
Input: Đồ thị, trạng thái bắt đầu, trạng thái đích đến, hàm heuristic
Output: Lời giải (Đường đi) or thất bại tìm kiếm
Bước 1:

- Khởi tạo hai tập rỗng open set và explored set
- Đưa trạng thái bắt đầu vào open set

Bước 2:
Khởi tạo vòng lặp trong khi open set vẫn chưa rỗng:
- Pop open set, lấy giá trị đỉnh trên đầu open set ra (đây là đỉnh có cost nhỏ nhất) được current node
- Nếu đỉnh này là goal thì trả về path và dừng thuật toán
- neighbors := những đỉnh kề với current node
- Khởi tạo vòng lặp với mỗi neighbor trong tập neighbors trên:

Nếu neighbor chưa nằm trong tập explored và không xuất hiện trong open set: Đưa neighbor vào explored, tính f = g + h, giá trị new cost := f, đưa neighbor vào open set với path và chi phí đó. Lặp lại bước 2.

Nếu neighbor xuất hiện trong open set: loại bỏ neighbor trong open set, tính f = g + h, giá trị new cost := f, đưa neighbor vào open set với path và chi phí đó. Lặp lại bước 2
Bước 3: Nếu ra khỏi vòng lặp mà chưa có đường di thì trả về thất bại cho bài toán

2.4.3 Đánh giá thuật toán

Đánh giá chung:
- Với A^* chắc chắn sẽ tìm được đường đi, nếu bài toán có tồn tại lời giải (đường đi) nhưng đường đi này không nhất thiết phải là đường đi tốt nhất do phụ thuộc vào hàm Heuristic.
- Bài toán có tìm được lời giải tối ưu được hay không là do cách chọn hàm Heuristic (Hàm Heuristic ở đây có thể có tính chất admissible heuristic, consistent heuristic). Khi hàm Heuristic chấp nhận được thì Tìm kiếm A* đầy đủ và tối ưu.

- Heuristic được tính chính xác từ trước: Dựa vào thông tin ước lượng khoảng cách từ cơ sở dữ liệu, data center mà tính toán ra được (Thường không hợp lý lắm vì rất nhiều vấn đề mà con

người vẫn chưa thể lưu trữ được hết xuống cơ sở dữ liệu)

- Heuristic dựa trên tri thức và kinh nghiệm: tri thức toán học về khoảng cách giữa hai điểm cho trước (với giả thiết là không có vật cản chắn giữa đường đi) như: Khoảng cách Manhattan, khoảng cách Chebyshev, khoảng cách Euclid, khoảng cách Diagonal, ...; tri thức về hướng, khoảng cách ước lượng nhờ tính theo đường chim bay, kinh nghiệm đời sống; ...

- Yếu điểm của thuật giải: Tốn không gian lưu trữ, vì cần lưu trữ nhiều các tập open set, explored set, path

2.4.3.1 Tính đầy đủ và tối ưu Thuật giải A^* sẽ cho ra kết quả đầy đủ và tối ưu nếu hàm heuristic $h(n)$ thỏa một số điều kiện nhất định (Phụ thuộc vào cách chọn hàm heuristic).
Với đồ thị hữu hạn, trọng số không âm, A^* đảm bảo được sự kết thúc và đầy đủ, tức là nó sẽ cho ra được lời giải (đường đi từ trạng thái bắt đầu đến trạng thái đích) nếu tồn tại.
Với đồ thị vô hạn, hệ số phân nhánh của mỗi nút hữu hạn và chi phí mỗi cạnh là một số dương, thuật giải vẫn có thể đưa ra được lời giải (đường đi từ trạng thái bắt đầu đến trạng thái đích) nếu tồn tại.

2.4.3.2 Độ phức tạp về thời gian Độ phức tạp về thời gian của A* phụ thuộc vào cách chọn hàm Heuristic hay chính là h(n). Hàm Heuristic tốt và tối ưu thì thuật toán đạt hiệu năng cao.Trong trường hợp xấu nhất của không gian tìm kiếm không bị giới hạn, số lượng nút được mở rộng là cấp số nhân theo chiều sâu d của lời giải (đường đi ngắn nhất): $O(b^d)$, trong đó b là hệ số phân nhánh

2.4.3.3 Độ phức tạp về không gian Độ phức tạp về không gian của A* gần giống như các thuật toán khác, vì tất cả mọi thứ như đường đi, chi phí cần được lưu trữ, và việc tính toán các hàm heuristic cũng phức tạp, nên sẽ rất tốn bộ nhớ.

2.5 Tìm kiếm tối ưu kiểu tham lam - Greedy Best First Search

2.5.1 Ý tưởng

Tìm kiếm tối ưu kiểm tham lam mở rộng nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến một lời giải một cách nhanh nhất. Lời giải trên nếu tồn tại có thể không là một lời giải "tốt nhất"
Đánh giá các trạng thại thông qua hàm Heuristic

$$f(n) = h(n)$$

Ta thấy, đây là một trường hợp của A* khi hàm g(n) = 0 (hằng 0)

2.5.2 Đánh giá thuật toán

2.5.2.1 Tính hoàn và tối ưu Vì đánh giá dựa vào hàm Heuristic nên ta sẽ tìm được lời giải cho bài toán, lời giải này không nhất thiết phải là một lời giải tốt nhất (hay một đường đi ngắn nhất), nó cố gắng mở rộng nút gần nhất với mục tiêu, với lý do rằng điều này có khả năng dẫn đến một giải pháp nhanh chóng nên nó không hoàn chỉnh và tối ưu [According to the book Artificial Intelligence: A Modern Approach (3rd edition), by Stuart Russel and Peter Norvig, specifically, section 3.5.1 Greedy best-first search (p. 92)]

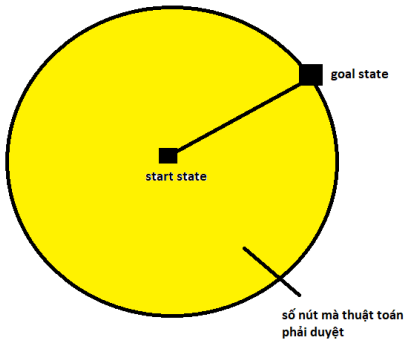
2.5.2.2 Độ phức tạp về thời gian Khi hàm đánh giá chính là hàm Heuristic, nó phụ thuộc vào cách chọn hàm Heuristic hay chính là h(n). Hàm Heuristic tốt và tối ưu thì thuật toán đạt hiệu năng cao, và nó sẽ gần giống với BFS

2.5.2.3 Độ phức tạp về không gian Khá tương tự với độ phức tạp về không gian của thuật giải A*.

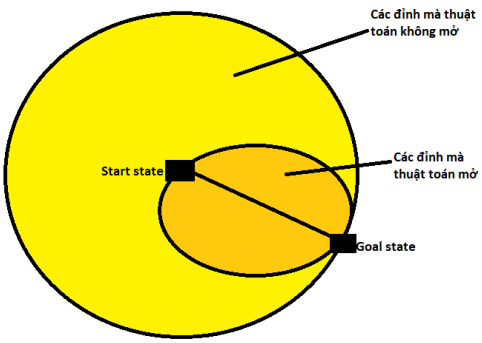
3 Hai thuật toán UCS và A^*

Bảng so so sánh sự khác biệt giữa hai thuật toán

Tiêu chí đánh giá	Uniform-cost Search	A* Search
Chiến lược	Sử dụng chiến lược tìm kiếm mù (không có thông tin), không có hàm đánh giá (tương tự như A* khi hàm h(n) = 0)	Sử dụng chiến lược tìm kiếm m có thông tin (kinh nghiệm), có sử dụng hàm đánh giá (f(n) = g(n) + h(n)
Chi phí	Được tính từ trạng thái bắt đầu đến trạng thái hiện tại	Tổng hai chi phí đi từ trạng thái gốc đến trạng thái hiện tại và một ước lượng chi phí từ trạng thái hiện tại đến trạng thái đích
Độ phức tạp theo thời gian	$O(b^{1+[\frac{C*}{e}]})$	$O(\|E\|) = O(b^d)$
Độ phức tạp theo không gian	$O(b^{1+[\frac{C*}{e}]})$	$O(\|E\|) = O(b^d)$
Danh sách các node chờ duyệt	Hàng đợi ưu tiên	Hàng đợi ưu tiên
Khả năng tìm nghiệm	Có nhưng một vài trường hợp bị vướng vòng lặp vô hạn	Có thể đưa ra nghiệm tốt
Tối ưu	Có vì chắc chắn tìm được lời giải tốt nhất nếu có tồn tại lời giải	Tìm đường lời giải tốt không nhất thiết phải là lời giải tốt nhất



Hình 3: Uniform Cost Search

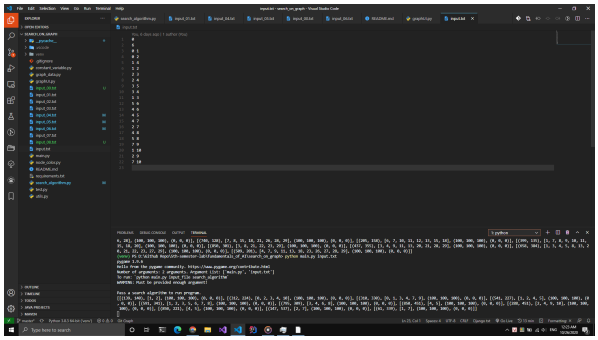


Hình 4: A* Search

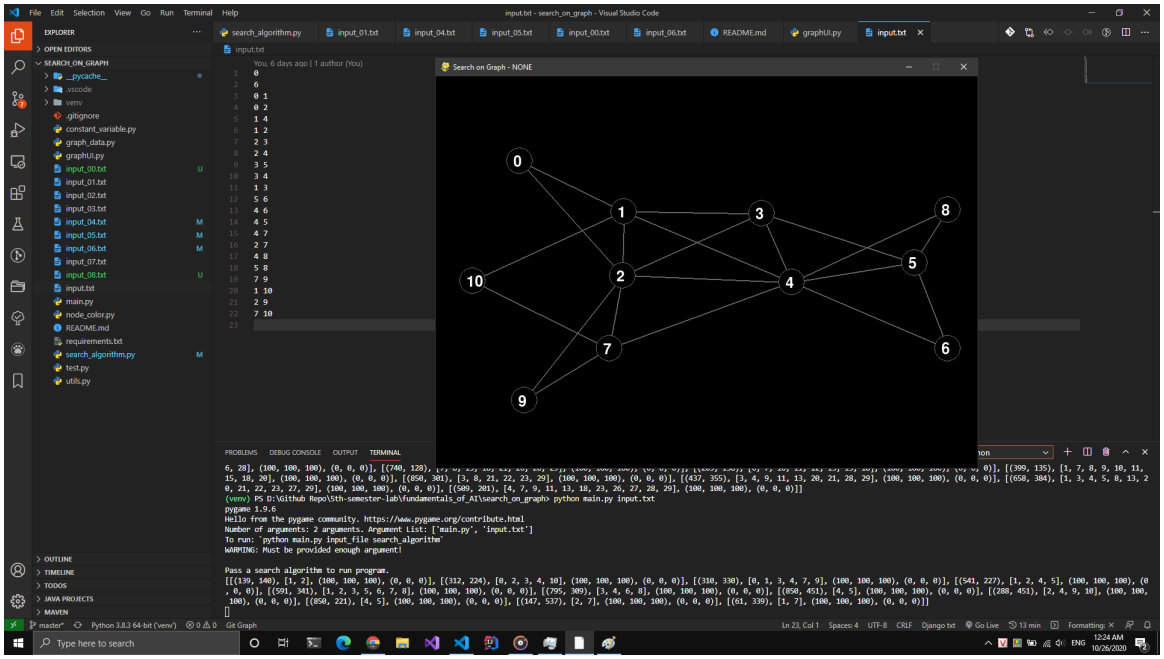
4 Demo lab với các testcase

4.1 Test case

4.1.1 Test case khi lúc nhận đồ án

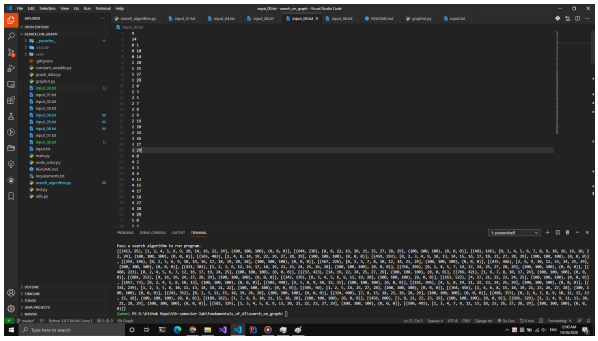


Hình 5: Testcase mẫu - Kích thước 11 đỉnh

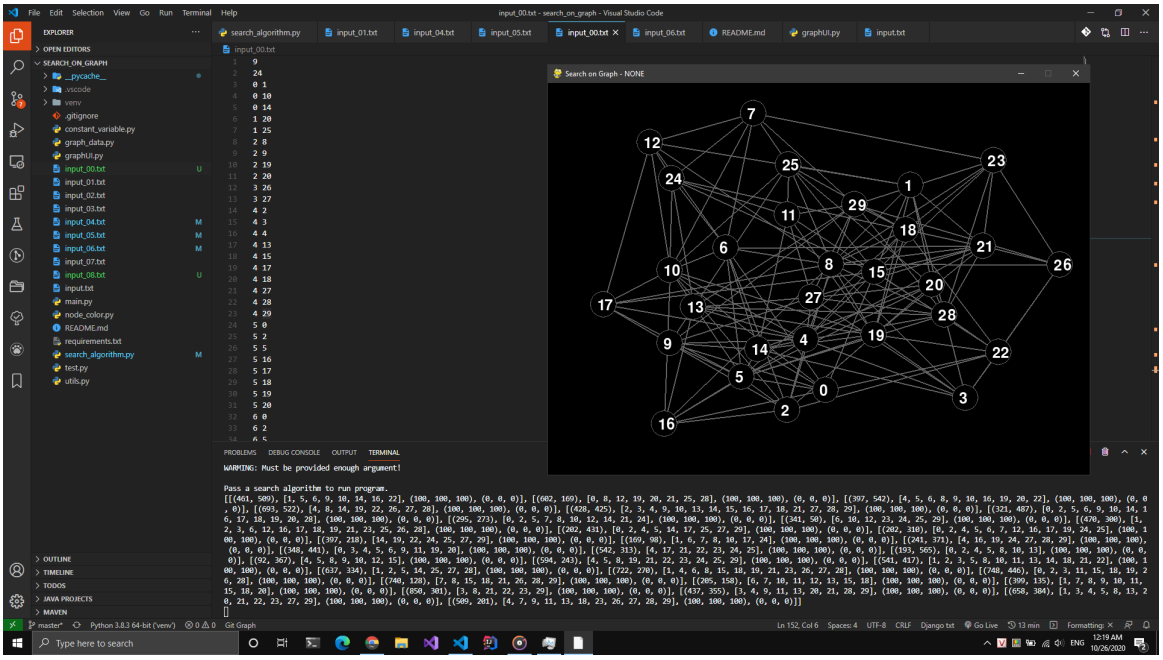


Hình 6: Testcase mẫu - Test hình dạng đồ thị

4.1.2 Test case 00: File input_00.txt



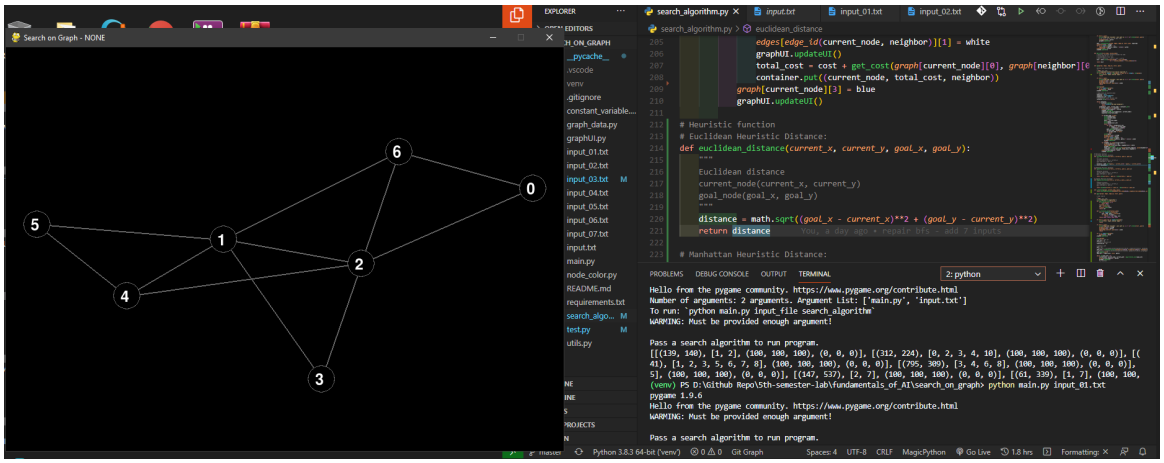
Hình 7: Testcase 00



Hình 8: Testcase 00 - Kích thước 20 đỉnh - Test hình dạng đồ thị

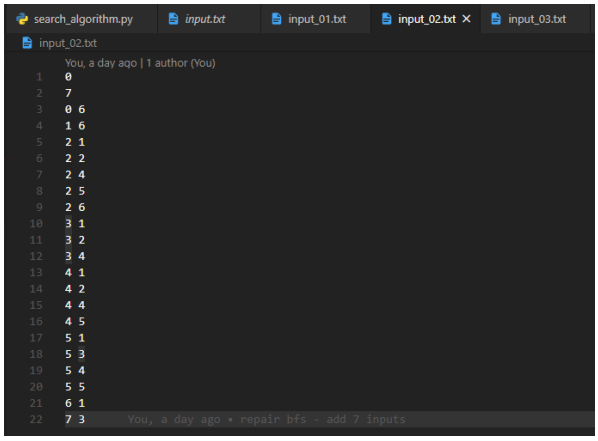
4.1.3 Test case 01: File input_01.txt

Hình 9: Testcase 01

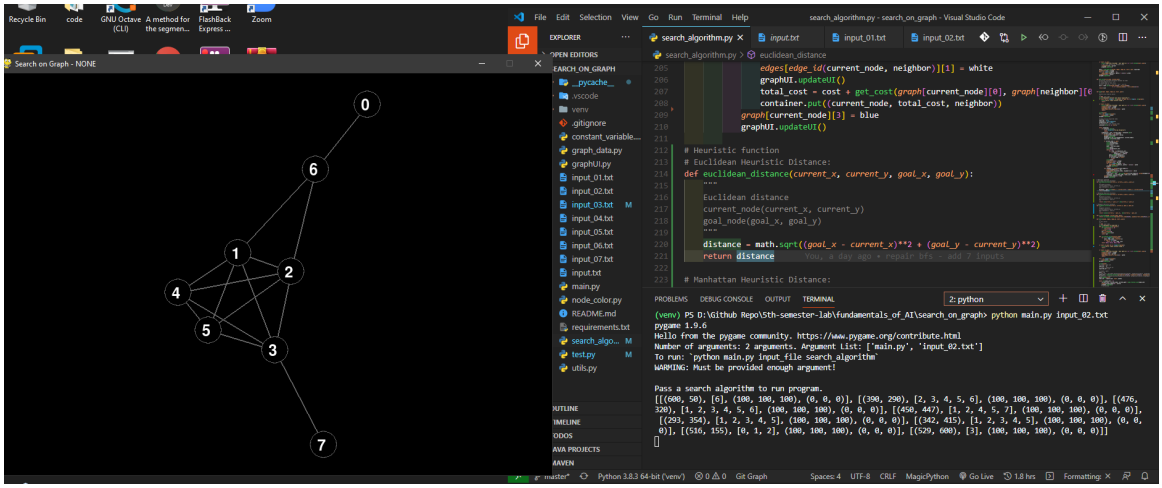


Hình 10: Testcase 01 - Kích thước 7 đỉnh - Test hình dạng đồ thị

4.1.4 Test case 02: File input_02.txt

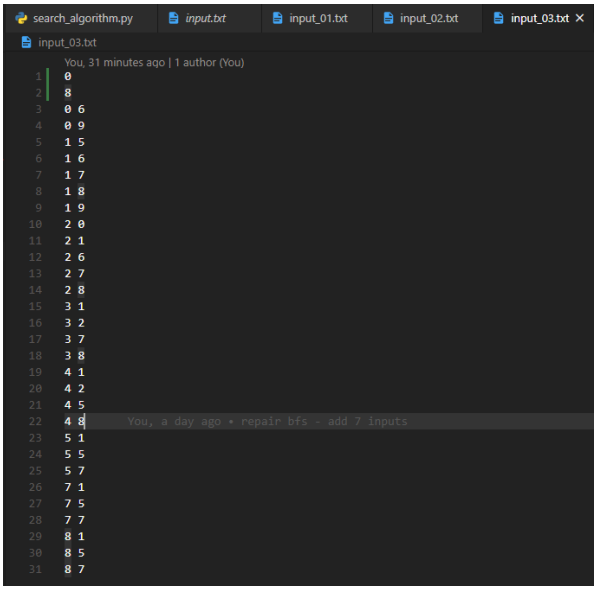


Hình 11: Testcase 02

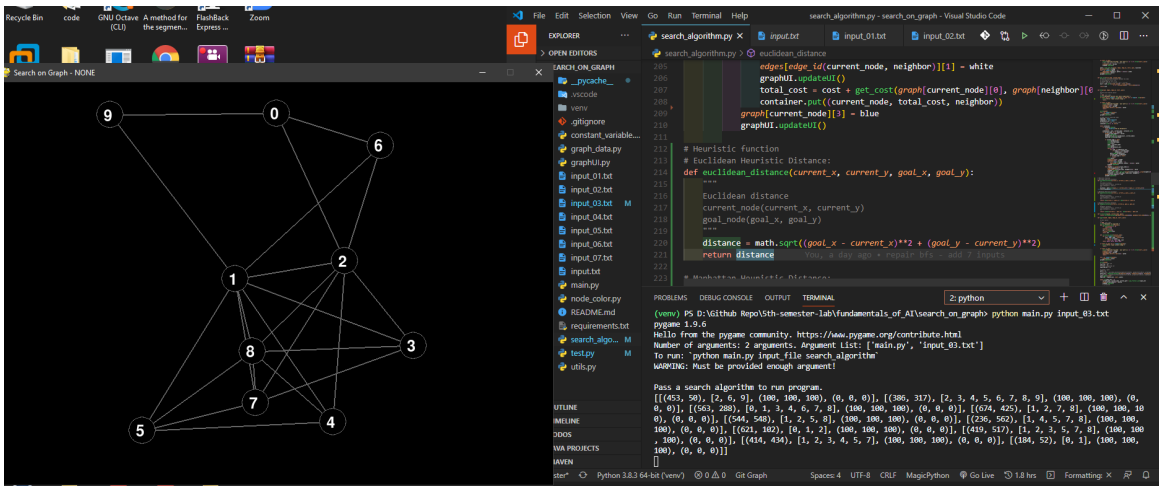


Hình 12: Testcase 02 - Kích thước 8 đỉnh - Test hình dạng đồ thị

4.1.5 Test case 03: File input_03.txt

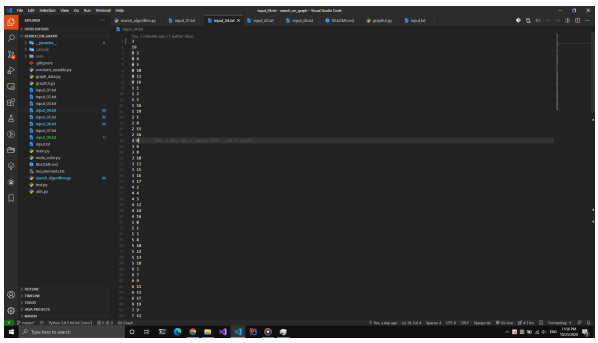


Hình 13: Testcase 03

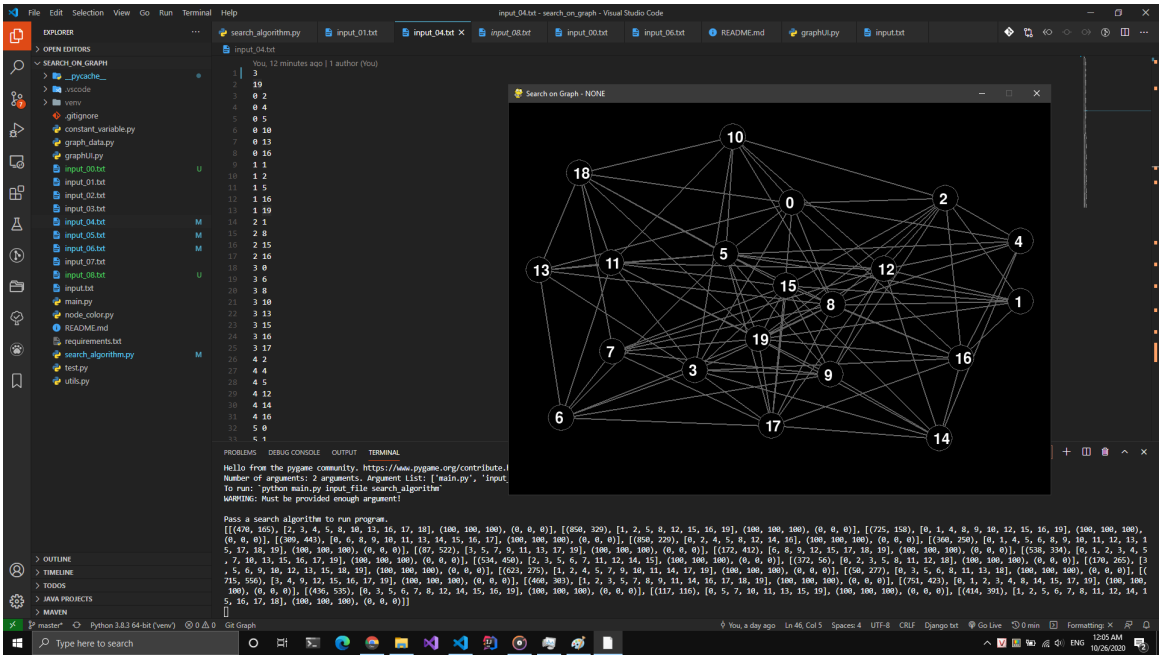


Hình 14: Testcase 03 - Kích thước 10 đỉnh - Test hình dạng đồ thị

4.1.6 Test case 04: File input_04.txt



Hình 15: Testcase 04



Hình 16: Testcase 04 - Kích thước 20 đỉnh - Test hình dạng đồ thị

Các video kiểm thử với từng thuật toán BFS, DFS, UCS, A Star, Greedy Best-First Search chứa trong thư mục video của đồ án

5 Tài liệu tham khảo

[1] Lê Hoài Bắc, Tô Hoài Việt. Giáo trình Cơ sở Trí tuệ Nhân tạo. Nhà xuất bản Khoa học và Kỹ thuật. Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Tp. Hồ Chí Minh.
[2] Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. (Third Edition). 2010. Publisher Prentice Hall
[3] Github: TheAlgorithms/Python