ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

CHỦ ĐỀ: SEARCH IN GRAPH

Môn: Nhập môn trí tuệ nhân tạo

Sinh viên: Trần Ý Văn - 21120360

Giảng viên hướng dẫn:

Giảng viên lý thuyết: Lê Hoài Bắc

Giảng viên thực hành: Nguyễn Bảo Long

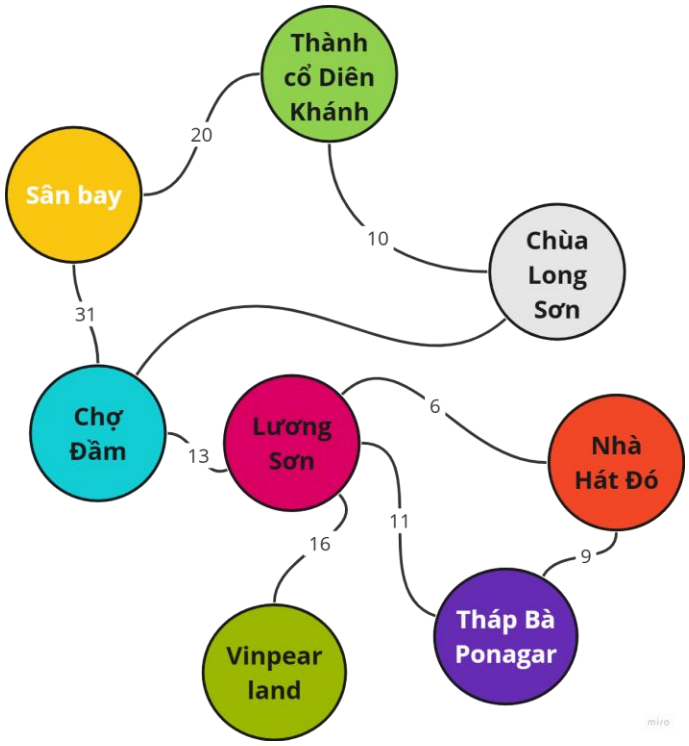
Thành phố Hồ Chí Minh – 2023

Contents

A. Các bài toán tìm kiếm (Search Problems)

I. Đặt vấn đề

- Adam là 1 du khách vừa đến Việt Nam. Hiện tại, anh ấy vừa đáp xuống sân bay Cam Ranh ở tỉnh Khánh Hòa. 1 người dân địa phương đã vẽ cho anh ấy 1 tấm bản đồ để anh ấy thuận tiện đi dạo quanh thành phố này.
- Bản đồ có dạng như sau:

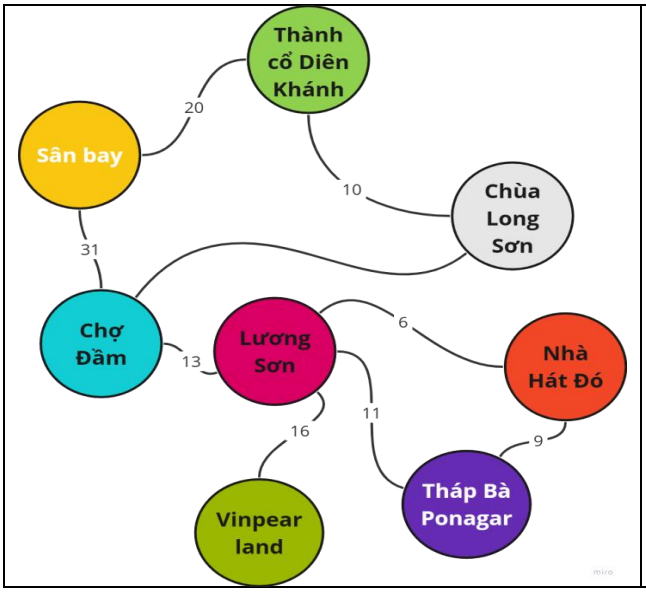


- Hiện tại anh ấy đang ở sân bay và anh ấy muốn tìm đường tới Tháp Bà Ponagar. Làm sao anh ấy có thể giải quyết vấn đề trên?

II. Các thành phần của bài toán tìm kiếm

- 1 bài toán tìm kiếm bao gồm không gian tìm kiếm (search space), trạng thái bắt đầu (start state), trạng thái kết thúc (goal state), successor function.
- Một tập các trạng thái có thể có của môi trường, nó được gọi là **không gian trạng thái (state space)** hay **không gian tìm kiếm (search space)**.
- **Trạng thái bắt đầu (start state)** mà tác nhân bắt đầu.
- Một tập gồm 1 hay nhiều trạng thái kết thúc (goal state), thường là 1 goal state, là đích đến hay mục tiêu cần đạt được trong bài toán tìm kiếm.
- Successor function

- 1 lời giải (**solution**) là 1 chuỗi thao tác (1 kế hoạch) để từ trạng thái bắt đầu có thể đạt được trạng thái kết thúc.
- **Successor function** là một hàm ánh xạ một trạng thái hiện tại sang một tập các trạng thái kế tiếp có thể đạt được từ trạng thái hiện tại. Mục đích của successor function là để khám phá không gian trạng thái của bài toán và tìm ra một lời giải hoặc một con đường tối ưu từ trạng thái ban đầu đến trạng thái đích.
- Mô hình hóa bài toán trên:

<table>
<tr>
<td> 
</td>
<td>+ Không gian trạng thái: Thành phố Nha Trang
+ Successor function:
Các con đường: Con đường nối 2 địa điểm kề nhau với chi phí là khoảng cách giữa chúng.
+ Trạng thái bắt đầu: Sân bay
+ Trạng thái kết thúc: Tháp Bà Ponagar</td>
</tr>
</table>

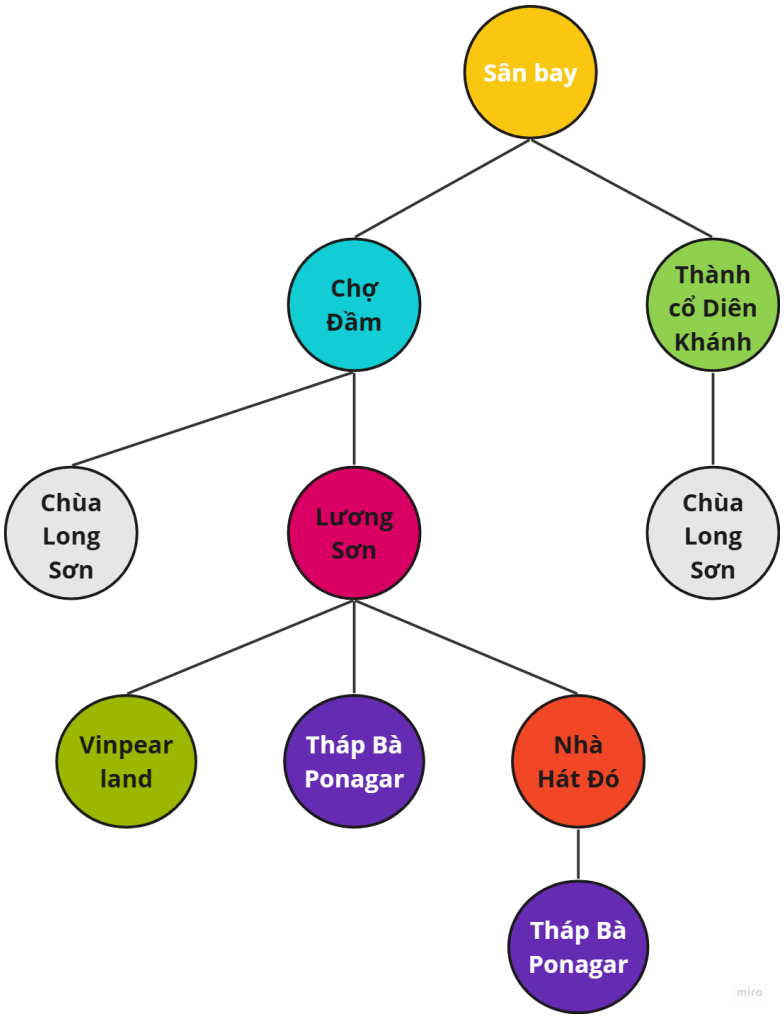
III. Giải quyết bài toán tìm kiếm

1. Đồ thị (Graph)

- Mô hình toán học của bài toán.
- Thể hiện được cấu trúc của bài toán.
- Hình ảnh bản đồ trên được vẽ dưới dạng đồ thị.

2. Cây tìm kiếm (Search Tree)

- Thể hiện sự truy vết trên đồ thị.
- Trạng thái có thể lặp lại trên cây tìm kiếm.
- 1 nhánh trên cây = 1 đường đi trên đồ thị.



3. Thuật toán tìm kiếm tổng quát.

```
General-search (problem, strategy)
  Initialize the search tree with the start state of the problem.
  Loop
    If there are no candidate states to explore return failure
    Choose a leaf node of the tree to expand next according to strategy
    If the node satisfies the goal condition return solution
    Expand the node and add all of its successors to the tree
  End loop
```

B. Uninformed Search Solutions

I. Định nghĩa

- Uninformed Search Algorithm hay thuật toán tìm kiếm không xác định sẽ không có manh mối nào về vấn đề để định hướng trong quá trình tìm kiếm. Thay vào đó, nó chỉ đơn giản duyệt không gian tìm kiếm 1 cách có hệ thống đến khi tìm được trạng thái đích.

- Ví dụ, từ sân bay có thể đi đến chợ Đầm hay Thành cổ nhưng Adam không phải người bản địa nên không biết đi đến đâu sẽ tốt hơn.

II. Breath-first Search (BFS)

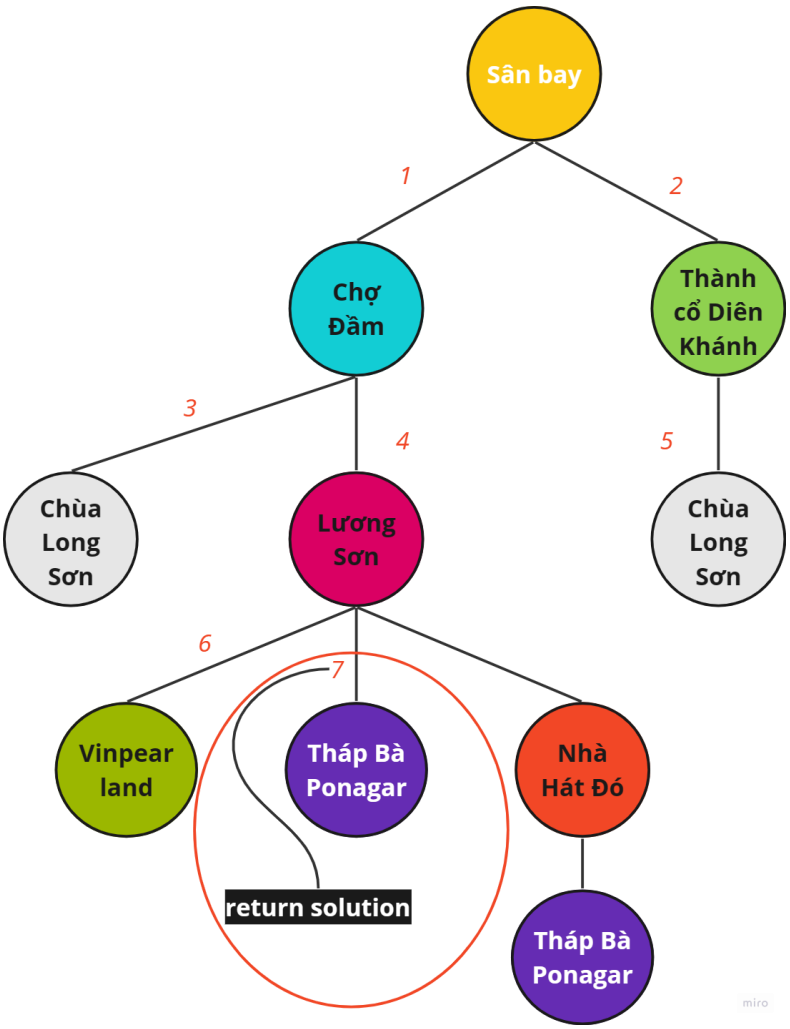
- Là một trong những thuật toán tìm kiếm cơ bản trên đồ thị.
- BFS có thể giải quyết các bài toán:
  - Tìm đường đi giữa các đỉnh trên đồ thị.
  - Xác định các thành phần liên thông của đồ thị.
  - Tìm đường đi ngắn nhất giữa các đỉnh trong đồ thị **không trọng số** hoặc **như nhau.**
- Ý tưởng
  - Giải quyết bài toán dựa trên ý tưởng "lập lịch".
  - Ưu tiên xử lý những đỉnh gặp được sớm nhất.
  - Những đỉnh gặp được sớm nhất là những đỉnh gần với đỉnh ban đầu nhất.

  ➔ Luôn đi đến các đỉnh theo con đường "ít cạnh nhất".
- Thuật toán thường được hiện thực cùng với cấu trúc dữ liệu queue. Những đỉnh được mở trước thì sẽ được duyệt trước.

```
BFS (problem)
  init_node <- problem.start_state
  opened <- a FIFO queue, with node as an element
  opened.push(init_node)
  closed <- a set which store closed nodes
  while (opened ≠ ∅) do
    cur_node <- opened.pop()
    closed.add(cur_node)
    if (cur_node == problem.goal_state) return solution
    for child_node in problem.get_children(cur_node) do
      if child_node not in closed and child_node not in opened
        opened.push(child_node)

  return failure
```

- Thể hiện:



- Những con số trên hình chỉ thứ tự duyệt cạnh đồ thị.
- Phân tích độ phức tạp thuật toán:
  - Thời gian: O(E*V), vì phải duyệt qua tất cả các cạnh, và trong mỗi lần gặp 1 đỉnh thì phải kiểm tra nó đã thuộc tập mở hay tập đóng chưa.
  - Không gian O(V), vì cần bộ nhớ cho tập mở và tập đóng.

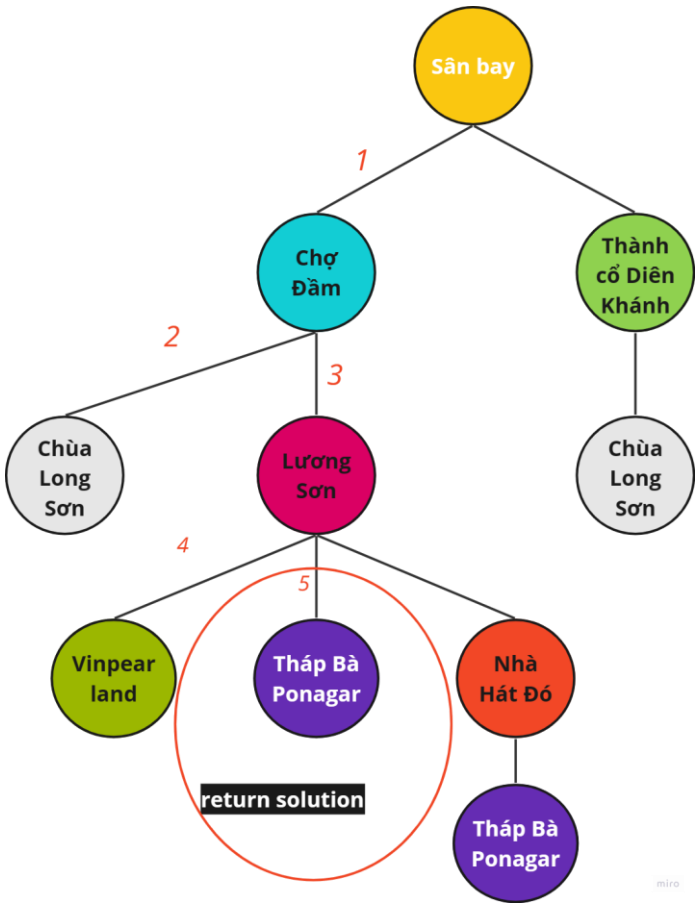
III. Depth-first Search (DFS)

- Thuật toán ưu tiên tiếp tục mở rộng các con đường đang có hiện tại để tìm đến đích. Nếu không có sẽ “quay lại” để tìm những con đường khác.
- Không chắc chắn cho ra đường đi ngắn nhất (đi qua ít cạnh nhất).
- Cũng hoạt động trên đồ thị không trọng số hoặc như nhau.
- Thuật toán thường được hiện thực với cấu trúc dữ liệu stack hoặc đệ quy, vì ưu tiên mở rộng con đường hiện tại (mới nhất).

```
DFS (problem)
  init_node <- problem.start_state
  opened <- a stack, with node as an element
  opened.push(init_node)
  closed <- a set which store closed nodes
  while (opened ≠ ∅) do
    cur_node <- opened.pop()
    closed.add(cur_node)
    if (cur_node == problem.goal_state) return solution
    for child_node in problem.get_children(cur_node) do
      if child_node not in closed and child_node not in opened
        opened.push(child_node)

  return failure
```

- Thể hiện trên đồ thị:



- Phân tích độ phức tạp: (tượng tự *BFS*)
  - Thời gian: O(E*V).
  - Không gian O(V).

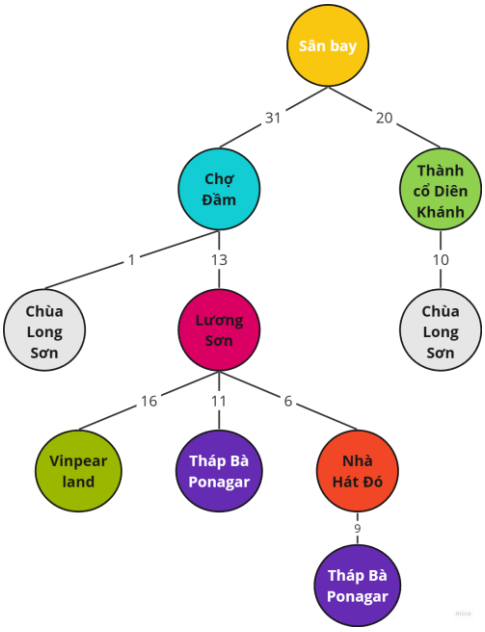
IV. Uniform-Cost Search (UCS)

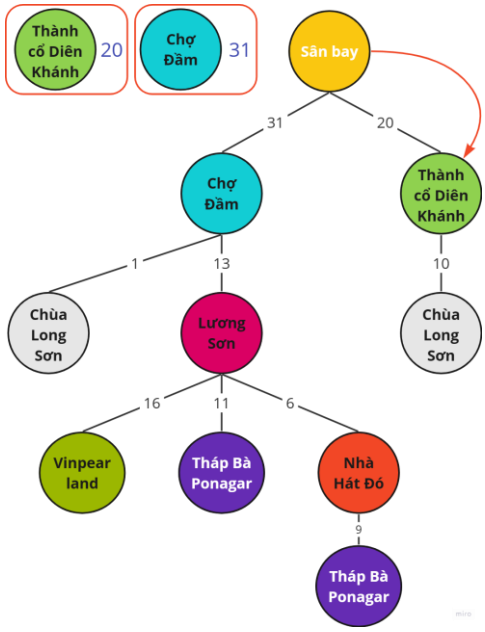
- UCS còn có 1 cái tên khác là Dijkstra.

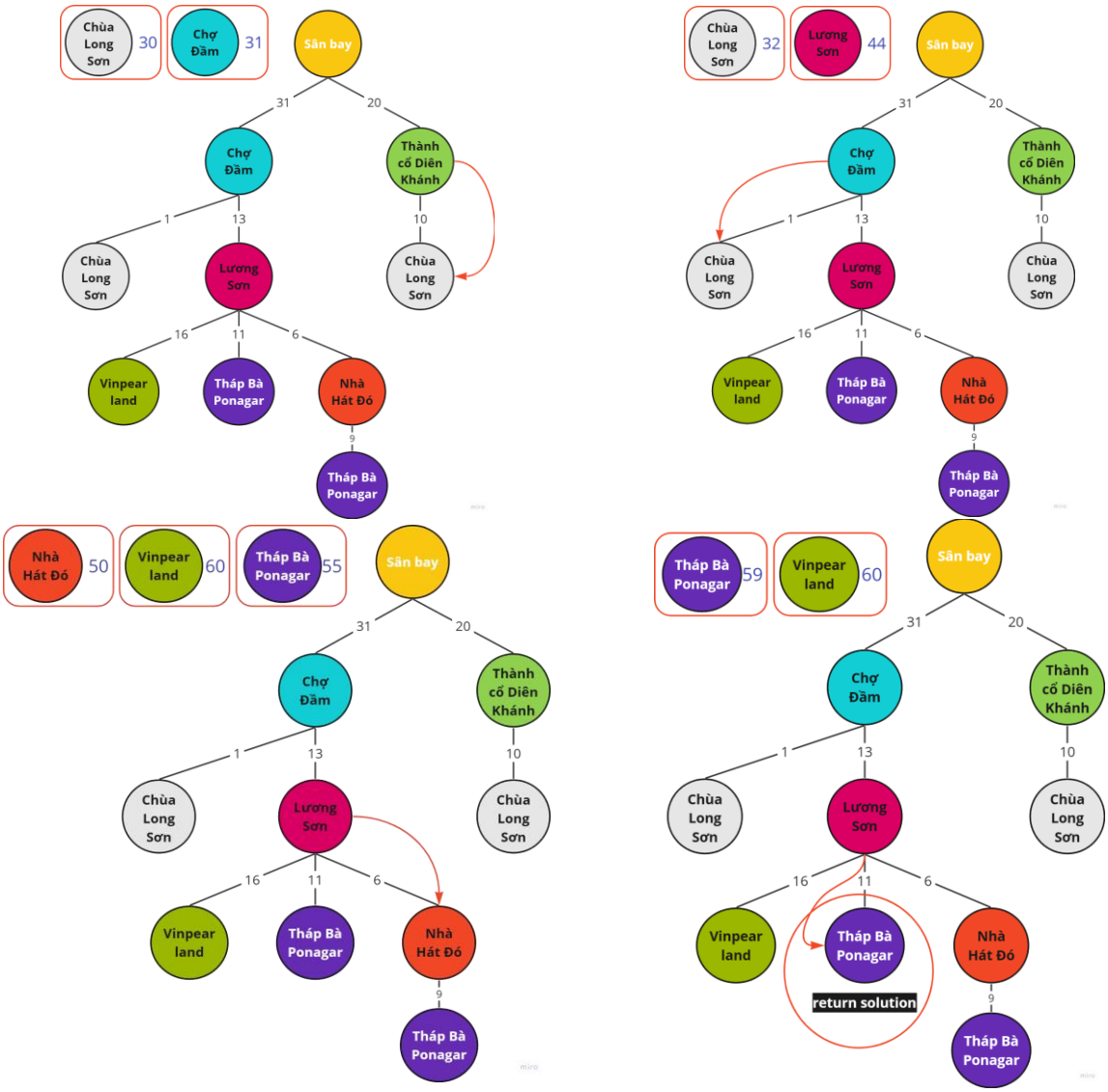
- Nó là thuật toán tìm đường đi có chi phí nhỏ nhất từ 1 đỉnh đến các đỉnh khác trong đồ thị có trọng số không âm.
- Ý tưởng:
  - Dựa trên tư tưởng tham lam.
  - Nhận xét 1: Vì chi phí của các cạnh không âm, nên càng đi qua nhiều cạnh thì tổng chi phí càng tăng.

  ➔ Từ 1 đường đi có chi phí X, không thể tồn tại cách mở rộng đường đi đến đỉnh khác mà được chi phí Y < X.
  - Nhận xét 2: trong các đỉnh có đường đi hiện tại, đường đi ngắn nhất là đường đi có khả năng mở rộng cao nhất và chắc chắn là đường đi không thể tối ưu thêm.

  ➔ Tại mỗi thời điểm, chọn đường đi đang có chi phí nhỏ nhất, cố định nó và mở rộng đường đi.







- Hiện thực:

```
UCS (problem)
  opened = priority_queue() #min priority queue
  opened.add((0, problem.start_id))
  cost = [inf]*problem.graph.length
  cost[problem.start_id] = 0

  while opened is not empty do
    curCost, curNode = opened.pop()
    if curCost != cost[curNode] continue
    if problem.is_goal(curNode) return solution
    for child in problem.get_children(curNode) do
      if cost[child] > cost[curNode] + problem.costOf(curNode, child) do
        opened.add((cost[child], child))

  return failure
```

- Phân tích độ phức tạp:
  - Thời gian: O(V + E*log(V))
  - Không gian: O(E)

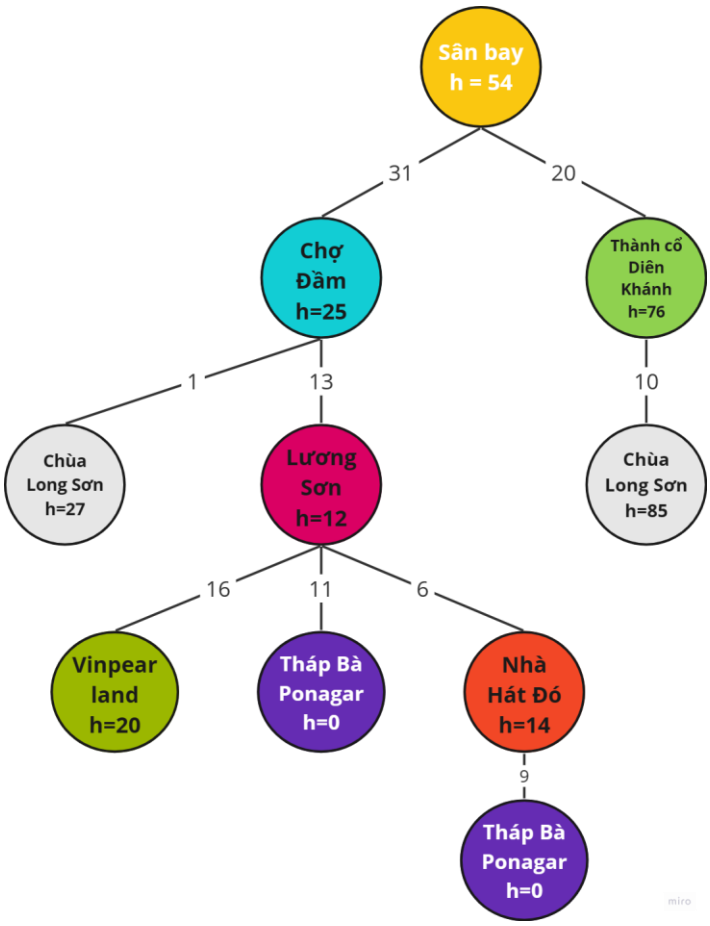
C. Informed Search Solutions

I. Định nghĩa

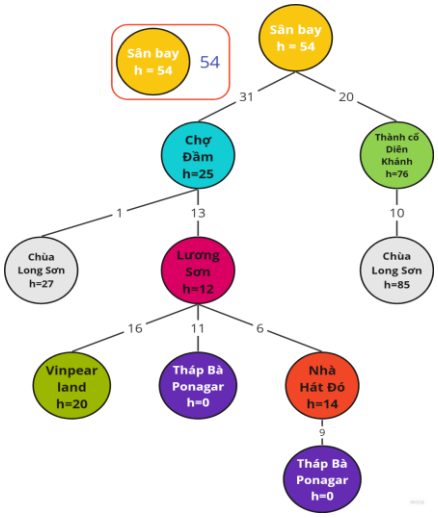
- Chiến lược này sử dụng các gợi ý (hint) về vị trí đích để tìm đường đi, có thể tìm lời giải hiệu quả hơn.
- Heuristic function: thường được thể hiện là h(n), với h(n) là ước lượng chi phí tốt nhất từ điểm n đi đến được đích.
- Ví dụ, 1 người dân bản địa họ có thể ước lượng khoảng cách từ Chợ Đầm đến Tháp Bà Ponagar là 25. Có thể nó không chính xác nhưng Adam có thể dựa vào gợi ý đó để tìm được đường tới đó.
- Kết quả của Informed Search phụ thuộc phần lớn vào hàm h(n), nếu người giải tìm được hàm h(n) càng tốt thì chi phí đi đến đích sẽ nhỏ hơn.

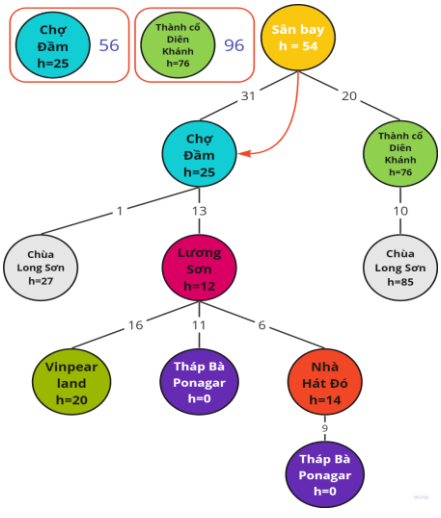
II. AStar Search (A*)

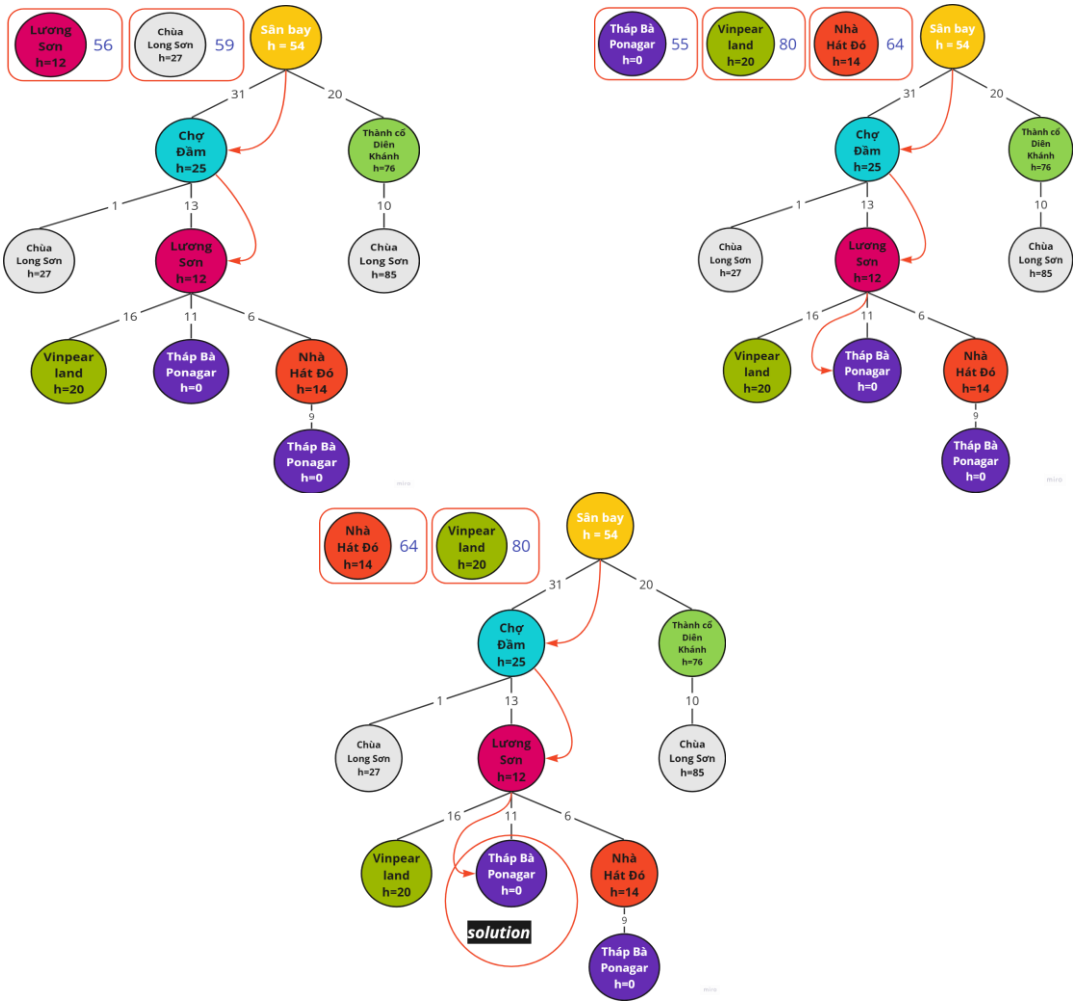
- Thuật giải A* sử dụng hàm đánh giá:
  f(n) = g(n) + h(n)
  Trong đó g(n) là chi phí đường đi từ start state đến node n.
- Do đó, f(n) là ước lượng chi phí của đường đi tốt nhất từ n đến đích.



- Từ đó, mục tiêu của A* là từ 1 state sẽ ưu tiên đi tới state con có giá trị f nhỏ nhất.







- Hiện thực:

```
AStar (problem)
  opened = priority_queue() #min priority queue
  opened.add(( h(problem.start_id), problem.start_id ))
  g = [inf]*problem.graph.length
  g[problem.start_id] = 0

  while opened is not empty do
    f, curNode = opened.pop()
    if f != g[curNode] + h(curNode) continue
    if problem.is_goal(curNode) return solution
    for child in problem.get_children(curNode) do
      old_f_child = g[child] + h(child)
      cur_f_child = g[curNode] + problem.costOf(curNode, child) + h(child)
      if old_f_child > cur_f_child do
        opened.add((cost[child], child))

  return failure
```

- Độ phức tạp:
  - Thời gian: O(V + E*log(V))
  - Không gian: O(E)

D. So sánh các giải thuật tìm kiếm

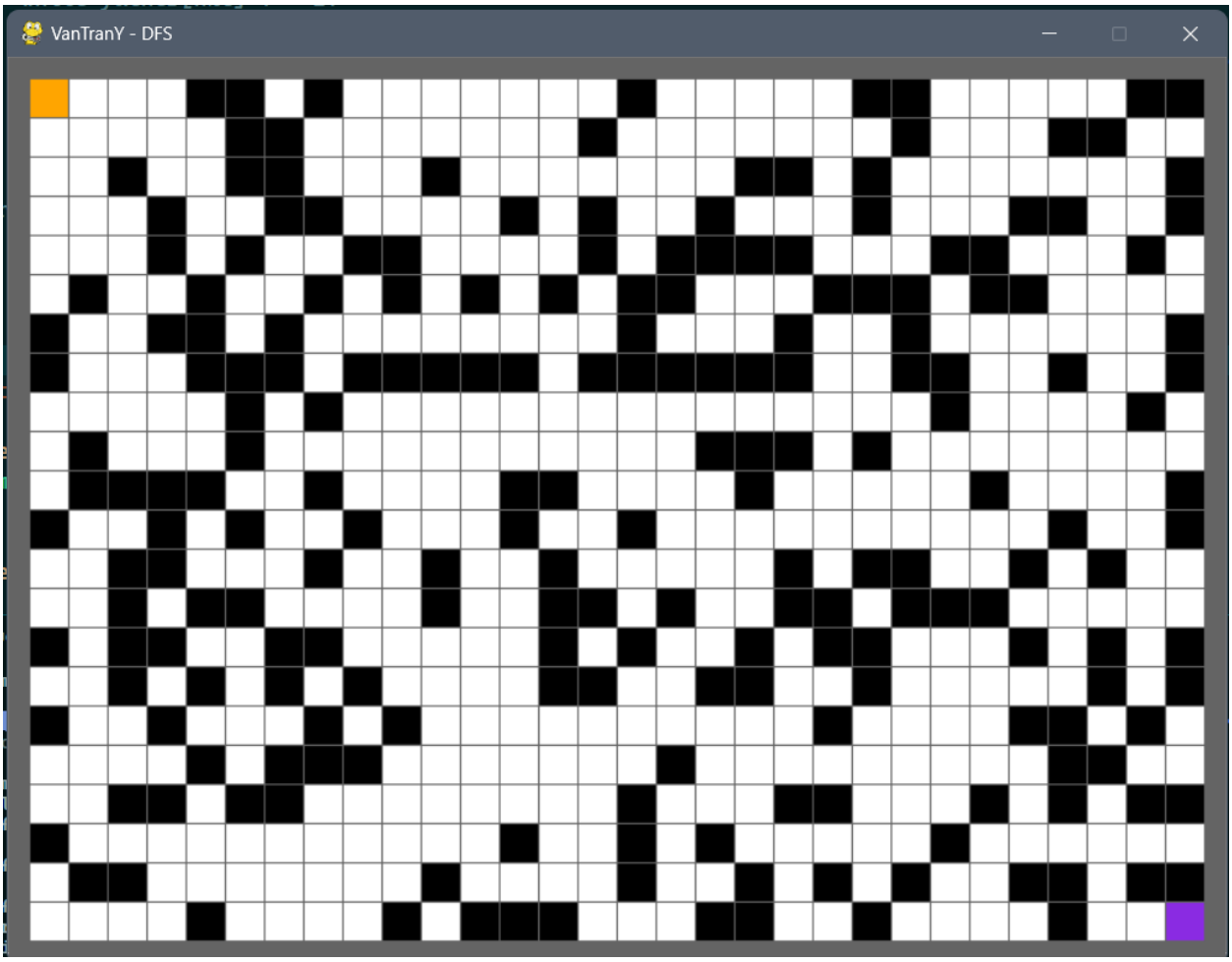
BFS	DFS	UCS	A*
Thường áp dụng trên đồ thị không trọng số.	Thường áp dụng trên đồ thị không trọng số.	Thường áp dụng trên đồ thị có trọng số.	Thường áp dụng trên đồ thị có trọng số.
Uninformed Search	Uninformed Search	Uninformed Search	Informed Search
Chỉ phụ thuộc vào cấu trúc đồ thị.	Chỉ phụ thuộc vào cấu trúc đồ thị.	Chỉ phụ thuộc vào cấu trúc đồ thị.	Kết quả phụ thuộc vào hàm h(n).
Time: O(V*E) Không gian: O(E)	Time: O(V*E) Không gian: O(E)	Time: O(V + E*log(V)) Không gian: O(E)	Time: O(V + E*log(V)) Không gian: O(E)

	UCS	Greedy Search	A* Search
Giống	Cả 3 đều là thuật toán tìm kiếm đường đi ngắn nhất giữa 2 trạng thái trên đồ thị Đều dùng chiến thuật tham lam (Greedy)		
Phân loại	Uninformed Search	Informed Search	Informed Search
Sử dụng thông tin về trạng thái đích	Không	Có	Có
Sự phụ thuộc đến kết quả	Hàm g(n)	Hàm h(n)	Cả h(n) và g(n)

E. Hiện thực các thuật toán (Implementation).

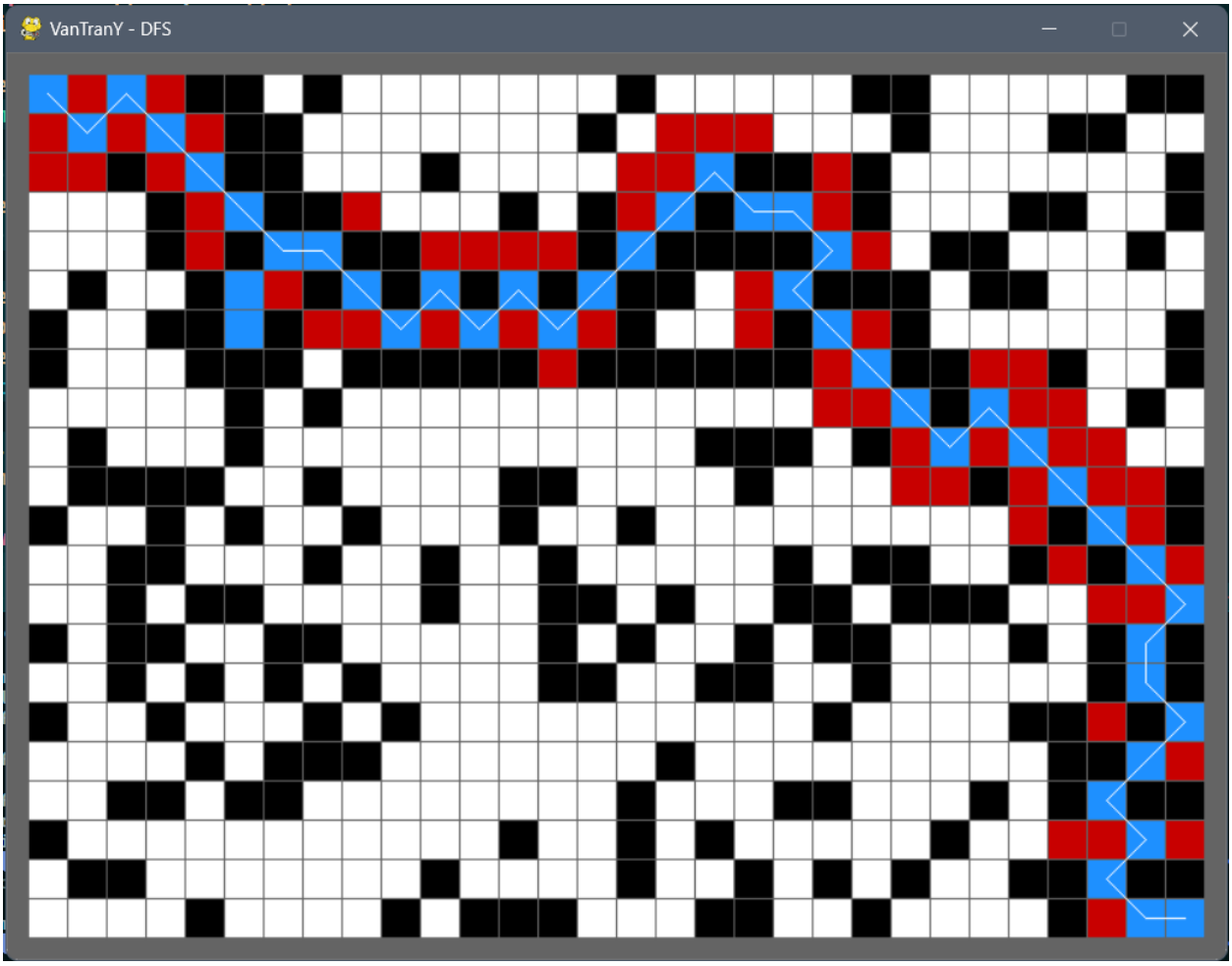
I. Bài toán

- Tìm đường đi từ vị trí ô xuất phát (cam) đến ô đích (tím)
- Không gian tìm kiếm: 1 ma trận với những ô trắng là ô có thể đi qua, ô đen là tường và không được phép đi.
- Trên 2 ô trắng kề nhau theo 8 hướng, người chơi có thể đi qua 2 những ô trắng ấy.



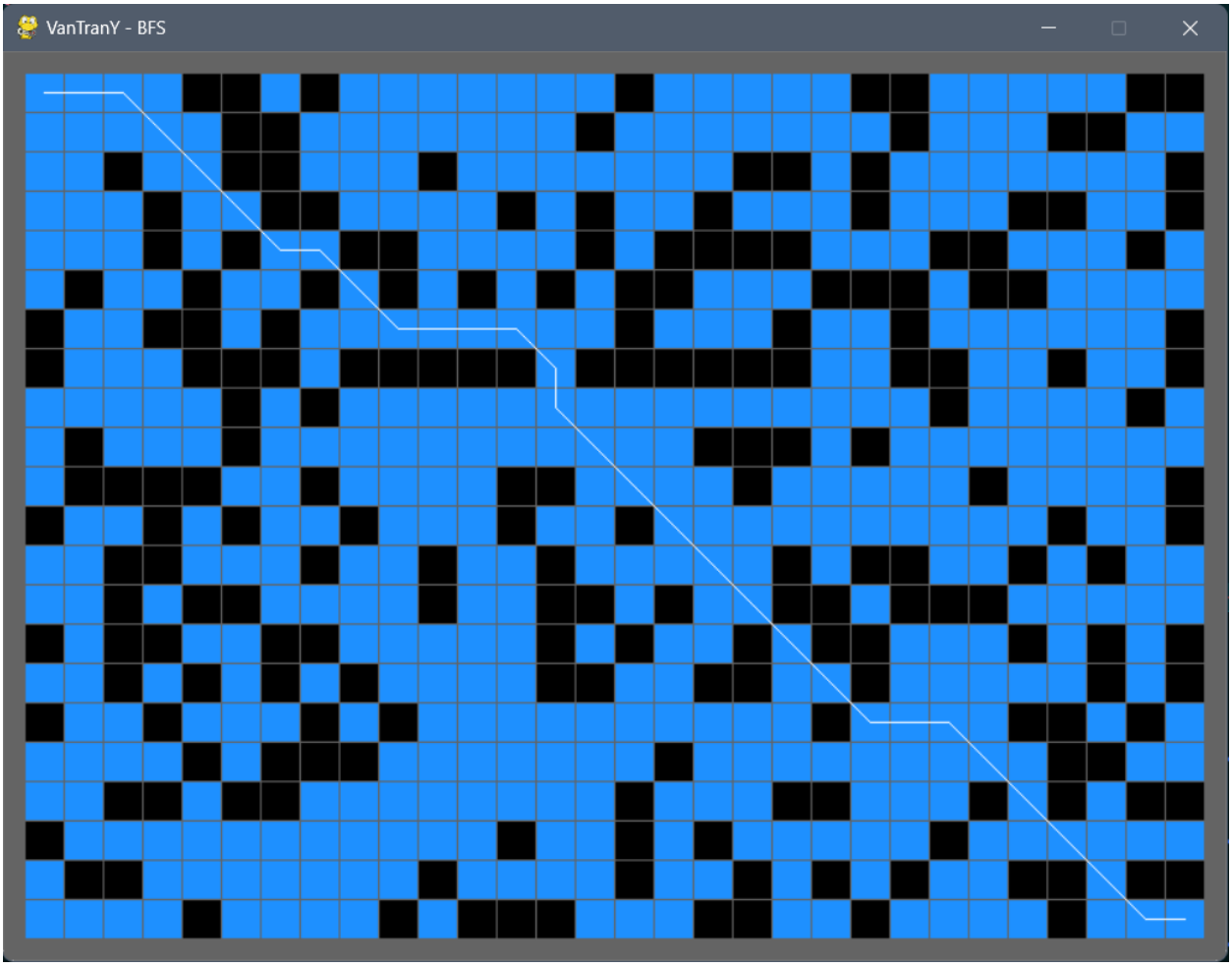
II. DFS

- Kết quả



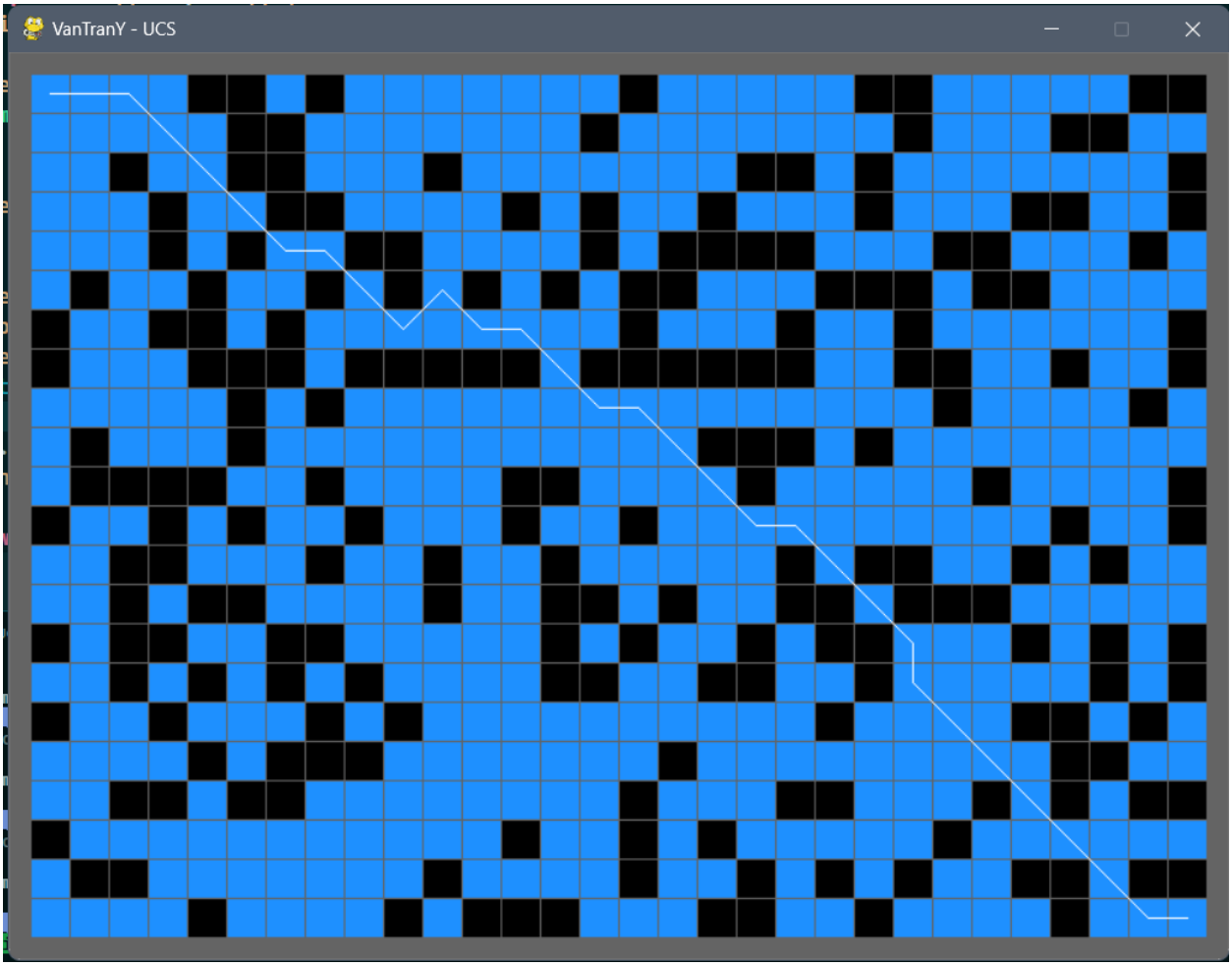
- Giải thích: Tại mỗi vị trí ô trắng, người chơi có xu hướng đi theo chiều sâu, tức là tiếp tục mở rộng con đường hiện tại đến khi không thể đi được nữa.
- Khi đến 1 vị trí mà không thể tiếp tục con đường đó thì người chơi sẽ đổi xoay qua hướng khác trong những hướng còn lại.
- Tiếp tục như thế đến khi chạm đến đích.
- Những ô xanh là những ô đã được đóng (đi qua hoàn toàn), còn những ô đỏ là những ô đang chờ để được đi qua.

III. BFS



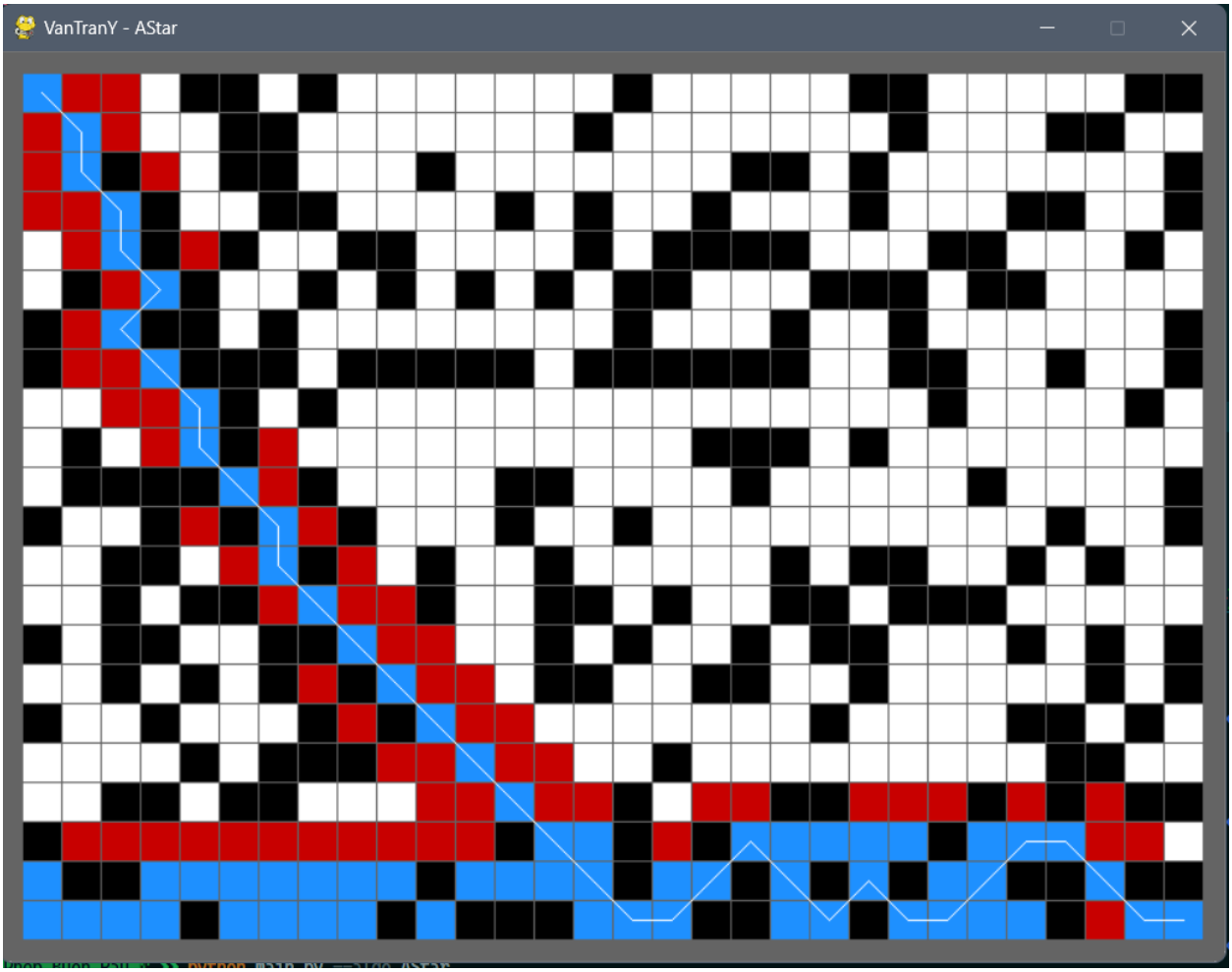
- BFS có xu hướng mở rộng đường đi theo chiều rộng, tức là từ 1 vị trí ô trắng, người chơi sẽ đi qua toàn bộ những ô trắng kề.
- Phương pháp này còn gọi là vết dầu loang. Đường đi sẽ được loang rộng đến khi chạm được đích.

IV. UCS



- UCS thường hoạt động hiệu quả trên đồ thị có trọng số, nhưng đây là ma trận, đường đi được hình thành từ 2 ô trắng kề nhau. Do đó có thể xem trọng số của cả đồ thị này bằng nhau, em giả định là 1.
- Vì cost giữa 2 ô kề nhau do đó từ vị trí 1 ô trắng, nó cũng sẽ có xu hướng đi đến những ô kề với nó trước. Hệ quả là xu hướng mở rộng ma trận cũng là vết dầu loang.

V. AStar



- Kết quả của A* sẽ phụ thuộc vào hàm h được chọn. Bản thân em chọn hàm h có dạng:

$$h(n) = g.get\_length() - 1 - n.id()$$

- Ví dụ:

id=0 h=3	id=1 h=2
id=2 h=1	id=3 h=0

length = 4

- Do đó, chẳng hạn người chơi đang ở vị trí ô id=0, ô tiếp theo được duyệt sẽ là id=3. Vì có h nhỏ nhất, khả năng nó gần đích cao hơn.
- Vì thế ở kết quả trên, người chơi có xu hướng đi tới ô right bottom hơn.
- Kết quả trên có thể không phải là con đường ngắn nhất nhưng có thể tìm được đường đi tới đích.

F. Nguồn tham khảo

- terminology - What is a successor function (in CSPs)? - Artificial Intelligence Stack Exchange
- Chuong 3. Tim kiem co ban.pdf (hust.edu.vn)
- Class3.pdf (pitt.edu)
- Artificial Intelligence: A Modern Approach, 4th US ed by Stuart Russell and Peter Norvig
- Tài liệu từ giảng viên.