ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

LAB 1 - SEARCH ALGORITHMS

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

Tp Hồ Chí Minh 2022

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGUYỄN MẠNH TƯỜNG

LAB 1 - SEARCH ALGORITHMS

Môn: Cơ Sở Trí Tuệ Nhân Tạo

MSSV: 20120619

Tên: Nguyễn Mạnh Tường

ĐỒ ÁN 1: CÁC THUẬT TOÁN TÍM KIẾM TRÊN ĐỒ THỊ

GIÁO VIÊN HƯỚNG DẪN

Thầy Nguyễn Bảo Long

Tp Hồ Chí Minh 2022

Mục lục

I. Tìm hiểu và trình bày thuật toán

1. Các thành phần của 1 bài toán tìm kiếm và cách giải.

1.1. Các thành phần

Một bài toán tìm kiếm có thể được định nghĩa bằng 5 thành phần:

- Trạng thái bắt đầu(Initial state)
- Mô tả các hành động (action) có thể thực hiện
- Mô hình di chuyển (transition model): mô tả kết quả của các hành động
- Kiểm tra đích (goal test): xác định một trạng thái có là trạng thái đích
- Một hàm chi phí đường đi (path cost) gán chi phí với giá trị số cho mỗi đường đi

1.2. Cách giải một bài toán tìm kiếm nói chung.

Một lời giải (solution) là một chuỗi hành động di chuyển từ trạng thái bắt đầu cho đến trạng thái đích. (Một lời giải tối ưu có chi phí đường đi thấp nhất trong số tất cả lời giải)

Các bước chính để giải 1 bài toán tìm kiếm:

- Xác định mục tiêu cần đạt đến (goal formulation)
    - Là một tập hợp của các trạng thái (đích)
    - Dựa trên: trạng thái hiện (của môi trường) và đánh giá hiệu quả hành động (của tác tử)
- Phát biểu bài toán (problem formulation)
    - Với một mục tiêu, các định các hành động và trạng thái cần xem xét
- Quá trình tìm kiếm (search process)
    - Xem xét các chuỗi hành động có thể
    - Chọn chuỗi hành động tốt nhất

Giải thuật tìm kiếm

- Đầu vào: một bài toán (cần giải quyết)
- Đầu ra: một giải pháp, dưới dạng một chuỗi các hành động cần thực hiện

1.3. Phân loại bài toán tìm kiếm (Uninformed Search và Informed Search)

Uninformed Search: còn được gọi là Blind search (Tìm kiếm điểm mù). Chiến lược tìm kiếm không dựa trên thông tin nào khác ngoài định nghĩa bài toán. Các thuật toán tìm kiếm điểm mù chỉ có khả năng sinh successor (các trạng thái có thể di chuyển được với một hành động duy nhất) và phân biệt trạng thái đích. Mỗi chiến lược tìm kiếm là một thể hiện (đồ thị/cây) của bài toán tìm kiếm tổng quát. Các thuật toán: DFS, BFS, UCS.

Informed Search: Tìm kiếm có định hướng. Các thuật toán có thông tin về trạng thái mục tiêu, Có khả năng tìm lời giải hiệu quả hơn so với các chiến lược tìm kiếm mù. Thông tin này được thu thập bằng một thứ gọi là phương pháp heuristic (ước lượng khoảng cách giữa trạng thái hiện tại so với trạng thái đích). Các thuật toán: A*, Greedy, …

2. Trình bày 4 thuật toán

2.1. DFS

Ý tưởng: Thuật toán DFS (Depth-first search – Duyệt theo chiều sâu) là thuật toán duyệt (tìm kiếm) trên cây hoặc đồ thị. Ý tưởng khởi đầu tại một nút gốc (hoặc một nút nào đó coi như gốc) và duyệt sâu theo một nhánh con của nút gốc. Khi không còn nút con để duyệt ta quay lui trở về nút cha, tiếp túc duyệt sâu các nút con còn lại của nút cha và lập lại cho tới khi tìm được kết quả hoặc hết nút để duyệt.

Mã giả:

```
Function Depth-Search(problem, Stack) returns a solution, or failure
    Stack ←  make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
        if Stack is empty then return failure;
        node = pop(Stack) ;
        if test(node,Goal[problem]) then return path(node,father);
        expand-nodes ← adjacent-nodes(node, Operators[problem]);
        push(Stack, expand-nodes );
```

foreach ex-node **in** expand-nodes

father(ex-node) = node;

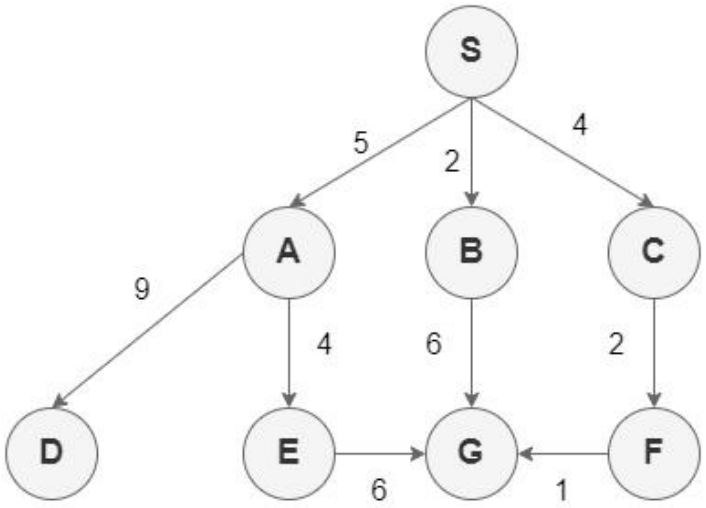
end

Đánh giá thuật toán:

- Tính đầy đủ: Phụ thuộc vào không gian tìm kiếm. Nếu không gian tìm kiếm là hữu hạn, thì DFS có tính đầy đủ. giải thuật không chắc chắn cho lời giải của bài toán trong trường hợp không gian trạng thái của bài toán là vô hạn.
- Tính tối ưu: Giải thuật DFS không cho ra lời giải tối ưu, số bước duyệt để đạt được giải pháp hoặc chi phí bỏ ra để đạt được nó là khá cao.
- Độ phức tạp về thời gian: O (b^m)
- Độ phức tạp về không gian: O (bm) kích thước không gian tuyến tính

Ví dụ minh họa đơn giản:

Đề bài: Duyệt độ thị bằng DFS tìm kiếm từ đỉnh S đến đỉnh G

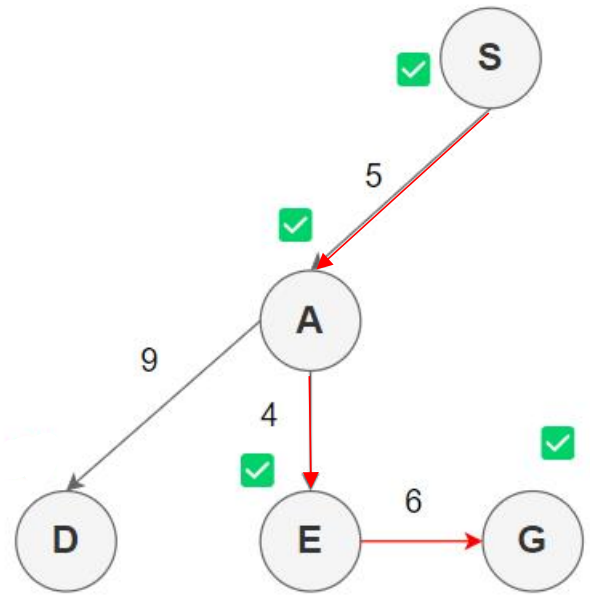


Bài làm:

Bảng mô tả các node được duyệt

Node	Stack
	S
S	A, B, C
A	D, E, B, C
D	E, B, C
E	G, B, C
G	

Hình vẽ cho kết quả tìm kiếm theo thuật toán DFS



2.2. BFS

Ý tưởng: Thuật toán BFS (Breadth-first search – Duyệt theo chiều sâu) là thuật toán duyệt (tìm kiếm) trên cây hoặc đồ thị. Ý tưởng từ một đỉnh, ta tìm các đỉnh kề rồi duyệt qua các đỉnh này. Tiếp tục tìm các đỉnh kề của đỉnh vừa xét rồi duyệt tiếp đến khi đi qua hết tất cả các đỉnh có thể đi.

Mã giả:

Function Breadth-Search(problem, Queue) **returns** a solution, or failure

Queue ← make-queue(make-node(initial-state[problem]));

father(initial-state[problem]) = empty;

while (1)

if Queue is empty then return failure;

node = pop(Queue) ;

if test(node,Goal[problem]) then return path(node,father);

expand-nodes ← adjacent-nodes(node, Operators[problem]);

push(Queue, expand-nodes);

foreach ex-node in expand-nodes

father(ex-node) = node;

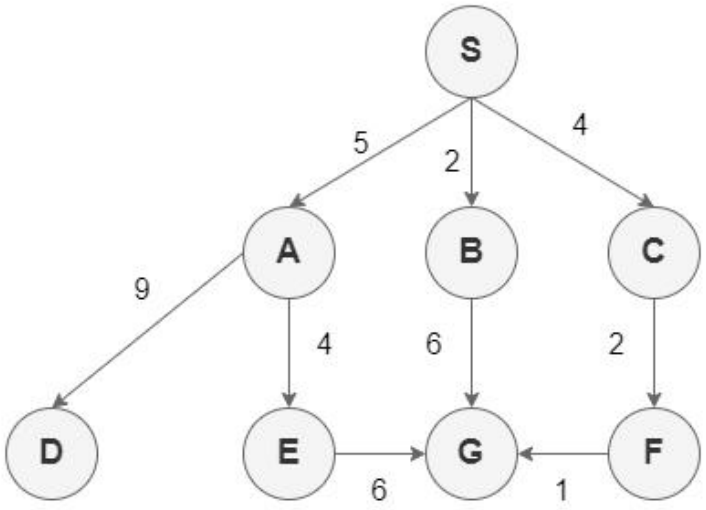
end

Đánh giá thuật toán:

- Tính đầy đủ: Thuật toán có tính đầy đủ, tức là nếu bài toán có lời giải, tìm kiếm theo chiều rộng đâm bảo tìm ra lời giải.
- Tính tối ưu: Có giải thuật tìm kiếm theo chiều rộng sẽ tìm ra lời giải với ít trạng thái trung gian nhất.
- Độ phức tạp thời gian: $1 + b + b^2 + ... + b^d$ (số vòng lặp khi gặp trạng thái đích) = O(b^d)
- Độ phức tạp về không gian: O(b^{d-1}) cho tập mở và O (b^d) cho biên.

Ví dụ minh họa đơn giản:

Đề bài duyệt đồ thị bằng BFS tìm đường đi từ đỉnh S đến đỉnh G

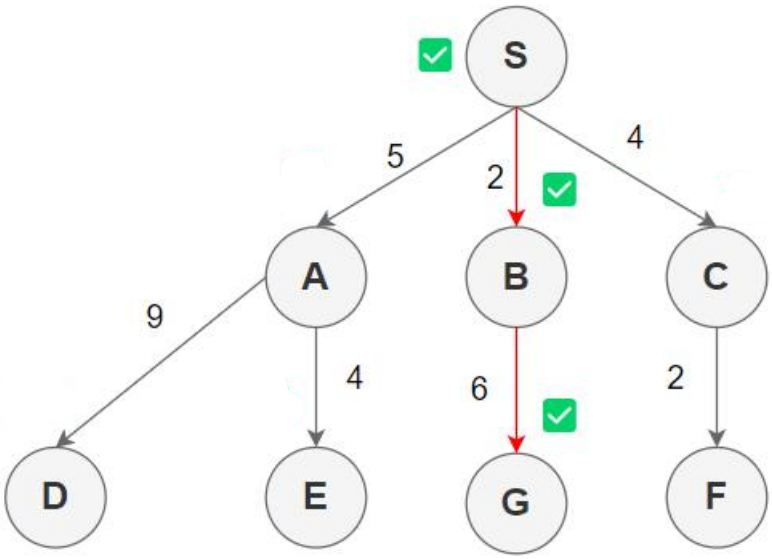


Bài làm:

Bảng mô tả các node được duyệt

Node	Queue
	S
S	A, B, C
A	B, C, D, E
B	C, D, E, G
C	D, E, G, F
D	E, G, F
E	G, F
G	F

Hình vẽ cho kết quả tìm kiếm BFS



2.3. UCS

Ý tưởng: Thuật toán UCS (Uniform-Cost-Search) tìm kiếm giá theo giá thành thống nhất. Ý tưởng tương tự như thuật toán BFS. Nhưng thay vì chọn nút nông nhất như tìm BFS để mở các nút tiếp thì UCS chọn nút cho chi phí thấp nhất để mở rộng.

Mã giả:

Function UNIFORM-COST-SEARCH(problem) **returns** a solution, or failure

if problem's initial state is a goal then return empty path to initial state

frontier ← a priority queue ordered by pathCost, with a node for the initial state

reached ← a table of {state: the best path that reached state}; initially empty

solution ← failure

while frontier is not empty and top(frontier) is cheaper than solution do

parent ← pop(frontier)

for child in successors(parent) do

s ← child.state

if s is not in reached or child is a cheaper path than reached[s] then

reached[s] ← child

add child to the frontier

if child is a goal and is cheaper than solution then

solution = child

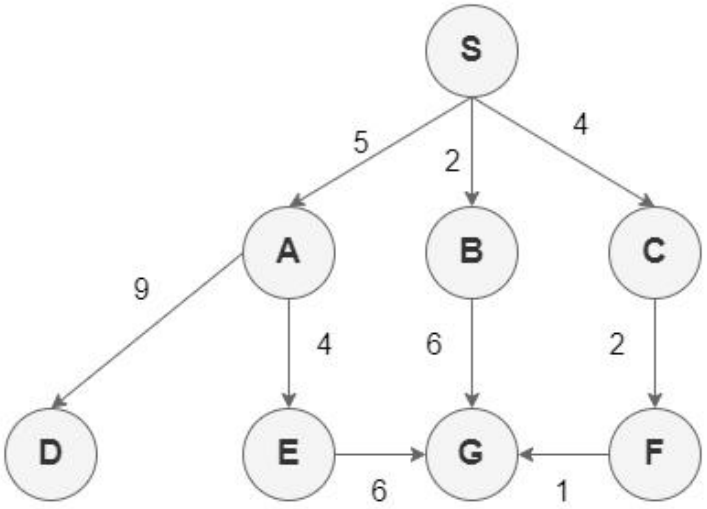
return solution

Đánh giá thuật toán:

- Với chi phí di chuyển thấp nhất là ε, C^* là chi phí lời giải tối ưu
- Tính đầy đủ: Có tính đầy đủ (nếu chi phí ở mỗi bước ≥ ε)
- Tính tối ưu: Có tính tối ưu (nếu các nút được xét theo thứ tự tăng dần về chi phí g(n))
- Độ phức tạp về thời gian: Phụ thuộc vào tổng số các nút có chi phí ≤ chi phí của lời giải tối ưu: O ($b^{1 + \lfloor C^*/\varepsilon \rfloor}$)
- Độ phức tạp về không gian: Phụ thuộc vào tổng số các nút có chi phí ≤ chi phí của lời giải tối ưu: O ($b^{1 + \lfloor C^*/\varepsilon \rfloor}$)

Ví dụ minh họa đơn giản:

Đề bài sử dụng thuật toán UCS để tìm đường đi từ đỉnh S đến đỉnh G.

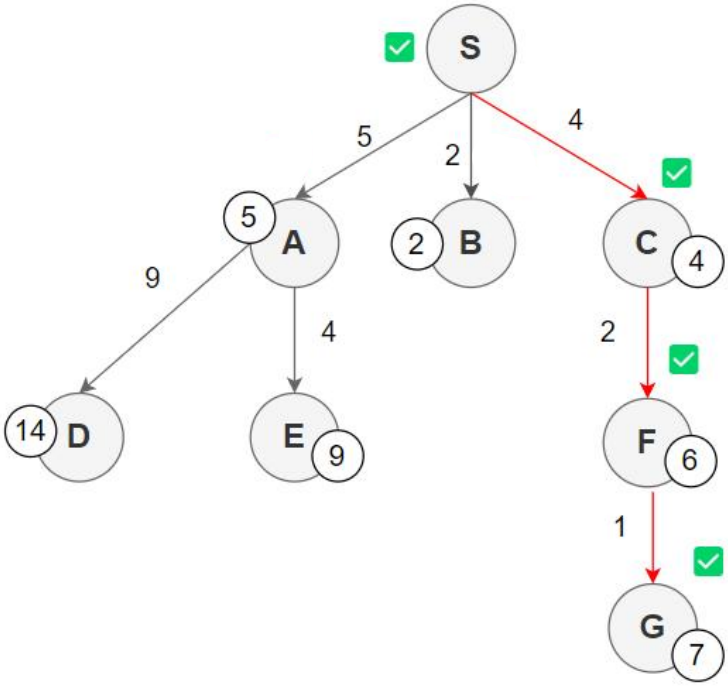


Bài làm:

Bảng mô tả các node được duyệt:

Node	Prority Queue
	S
S	B, C, A
B	C, A, G
C	A, F, G
A	F, G, D, E
F	G, D, E
G	

Hình vẽ mô tả kết quả:



2.4.Astar

Ý tưởng: A* (đọc là A sao) là dạng tìm kiếm Best-first Search phổ biến nhất. Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút

thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Ý tưởng trực quan xét bài toán tìm đường – bài toán mà A* thường được dùng để giải. A* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin (informed tìm kiếm thuật toán), nó chỉ xây dựng các tuyến đường "có vẻ" dẫn về phía đích. Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A* sử dụng một "đánh giá heuristic" về khoảng cách từ điểm bất kỳ cho trước tới đích.

Mã giả:

```
Function Astar-Search(start, end)
    open_list ← {start}
    closed_list ← {}
    g(start) ← 0
    h(start) ← heuristic_function(start,end)
    f(start) ← g(start) + h(start)
    while open_list.isEmpty():
        curr ← Node on top of open_list, with least f
        if curr == end then return curr
        open_list.remove(curr)
        closed_list.add(curr)
        for each n in child(curr)
            if n in closed_list
                continue
            cost ← g(curr) + distance(curr, n)
            if n in open_list and cost < g(n)
                open_list.remove(n)
            if n in closed_list and cost < g(n)
```

```
                    closed_list.remove(n)
            if n not in open_list and n not in closed_list
                    open_list.add(n)
                    g(n) ← cost
                    h(n) ← heuristic_function(n,end)
                    f(n) ← g(n) + h(n)
    return failure
end
```

Đánh giá thuật toán:

- Với chi phí di chuyển thấp nhất là ε, C^* là chi phí lời giải tối ưu
- Tính đầy đủ: Thuật toán có tính đầy đủ (nếu ε > 0 và không gian trạng thái hữu hạn
- Tính tối ưu: Thuật toán có tính tối ưu nếu heuristic hợp lý và nhất quán
- Độ phức tạp về thời gian: Cấp số mũ
- Độ phức tạp về không gian: Cấp số mũ
- Đề xuất heuristic: Các hàm heuristic đóng vai trò rất quan trọng. Hàm heuristic tốt đẩm bảo tính tối ưu cho những thuật toán như A*. Bên cạnh đó hàm heuristic tố cho phép hướng thuật toán theo phía lời giải, nhờ vậy giảm số trạng thái cần khảo sát và độ phực tạp của thuật toán.

  Các Hàm heuristic h(n) được xây dựng tùy thuộc vào bài toán cụ thể. Cùng một loại bài toán chúng ta có thể có rất nhiều hàm heuristic khác nhau. Chất lượng hàm heuristic ảnh hưởng rất nhiều đến quá trình tìm kiếm.

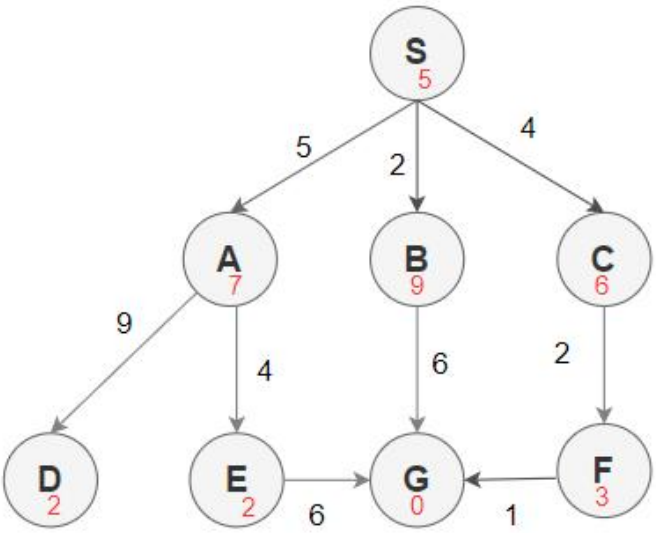
  Hàm heuristic h(n) được gọi là chấp nhận được khi:

  h(n) <= h*(n)

  Trong đó h*(n) là giá thành đường đi thực từ n đến node đích.

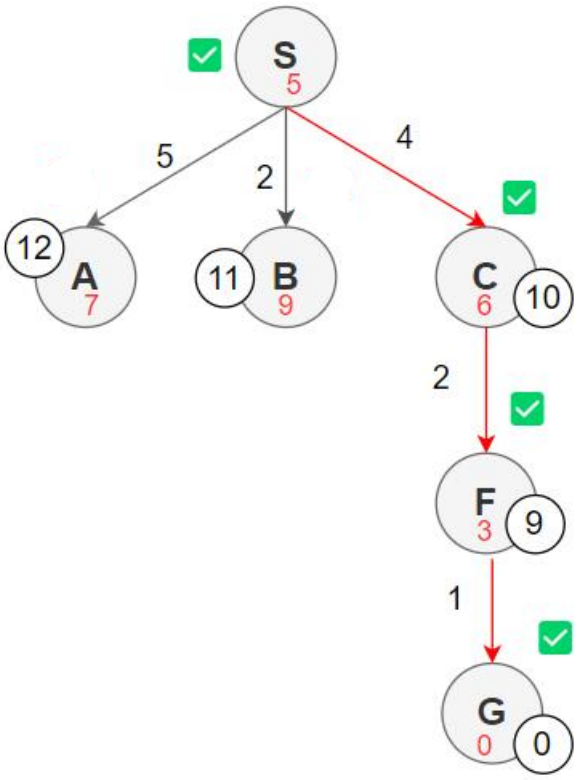
Ví dụ minh họa:

Đề bài sử dụng thuật toán A* tìm đường đi từ đỉnh S đến đỉnh G



Bài làm:

Hình vẽ minh họa đường đi của thuật toán A*



II. So sánh

1. So sánh sự khác biệt giữa UCS, Greedy và A*

Cơ sở so sánh	UCS	Greedy	A*
Đặc điểm	Thuật toán UCS chọn nút n có chi phí đường đi g(n) nhỏ nhất để mở rộng thay vì chọn nút nông nhất như BFS	Thuật toán Greedy Search là chọn trong tập nút biên nút có khoảng cách tới đích nhỏ nhất để mở rộng. Trong phương pháp này để đánh giá độ tốt của một nút ta sử dụng hàm heuristic (hàm ước lượng chi phí từ một nút đến đỉnh). Ở thuật thuật toán Greedy ta có f(n) = h(n)	Tương tự thuật toán Greedy. Nhưng thuật toán A* sử dụng hàm đánh giá f(n) với 2 thành phần, thành phần một là đường đi từ nút đang xét tới nút xuất phát, thành phần 2 là khoảng cách ước lượng tới đích. f(n) = g(n) + h(n)
Tính đầy đủ	Có tính đầy đủ	Không có tính đầy đủ do có khả năng tạo thành vòng lặp vô hạn ở một số nút.	Có tính đầy đủ nếu không gian trạng thái hữu hạn.
Tính tối ưu	Có tính tối ưu	Thuật toán không tối ưu	Thuật toán có tính tối ưu nếu heuristic hợp lý và nhất quán.
Độ phức tạp về thời gian	O ($b^{1+\lfloor C^*/\varepsilon \rfloor}$)	Trong trường hợp xấu nhất là O (b^m)	Cấp số mũ
Độ phức tạp	O ($b^{1+\lfloor C^*/\varepsilon \rfloor}$)	Trong trường hợp xấu nhất	Cấp số mũ

về không gian.		là O (b^m)	

2. So sánh sự khác biệt giữa UCS và Dijkstra.

- Thuật toán Dijkstra tìm đường đi ngắn nhất từ nút gốc đến mọi nút khác. Còn thuật toán UCS tìm kiếm các đường đi ngắn nhất về chi phí từ nút gốc đến nút mục tiêu. Thuật toán UCS là trường hợp riêng của thuật toán Dijkstra tập trung vào việc tìm kiếm một con đường ngắn nhất duy nhất đến một điểm kết thúc duy nhất thay vì con đường ngắn nhất đến mọi điểm.
- Thuật toán UCS thực hiện tìm kiếm bằng cách dừng lại ngay khi tìm thấy điểm kết thúc. Đối với thuật toán Dijkstra, không có trạng thái mục tiêu và quá trình xử lý vẫn tiếp tục cho đến khi tất cả các nút đã được xóa khỏi hàng đợi ưu tiên, tức là cho đến khi xác định được đường đi ngắn nhất đến tất cả các nút (không chỉ một nút mục tiêu).
- Thuật toán UCS có ít yêu cầu về không gian hơn, trong đó hàng đợi ưu tiên được lấp đầy dần dần trái ngược với thuật toán Dijkstra thêm tất cả các nút vào hàng đợi ngay từ đầu với chi phí vô hạn.
- Thuật toán Dijkstra cũng tốn nhiều thời gian hơn thuật toán UCS. Do yêu cầu về bộ nhớ.
- Thuật toán UCS thường được xây dựng trên cây trong khi Dijkstra được sử dụng trên đồ thị chung.
- Thuật toán Djikstra chỉ có thể áp dụng trong các đồ thị tường mình mà chúng ta biết tất cả các đỉnh và cạnh trong đó toàn bộ đồ thị được đưa ra làm đầu vào. UCS bắt đầu với đỉnh nguồn và dần dần đi qua các phần cần thiết của đồ thị. Do đó, nó có thể áp dụng cho cả đồ thị rõ ràng và đồ thị không tường minh (nơi các trạng thái / nút được tạo ra).
- Một sự khác biệt nhỏ khác giữa hai thuật toán này là các giá trị khoảng cách cuối cùng trên các đỉnh không thể truy cập được từ đỉnh nguồn. Trong thuật toán

Dijkstra, nếu không có đường đi giữa đỉnh nguồn S và đỉnh V thì giá trị khoảng cách của nó (dist [v]) là ∞. Tuy nhiên, trong thuật toán UCS, chúng ta không thể tìm thấy giá trị đó trong bản đồ khoảng cách cuối cùng, tức là dist [v] không tồn tại.

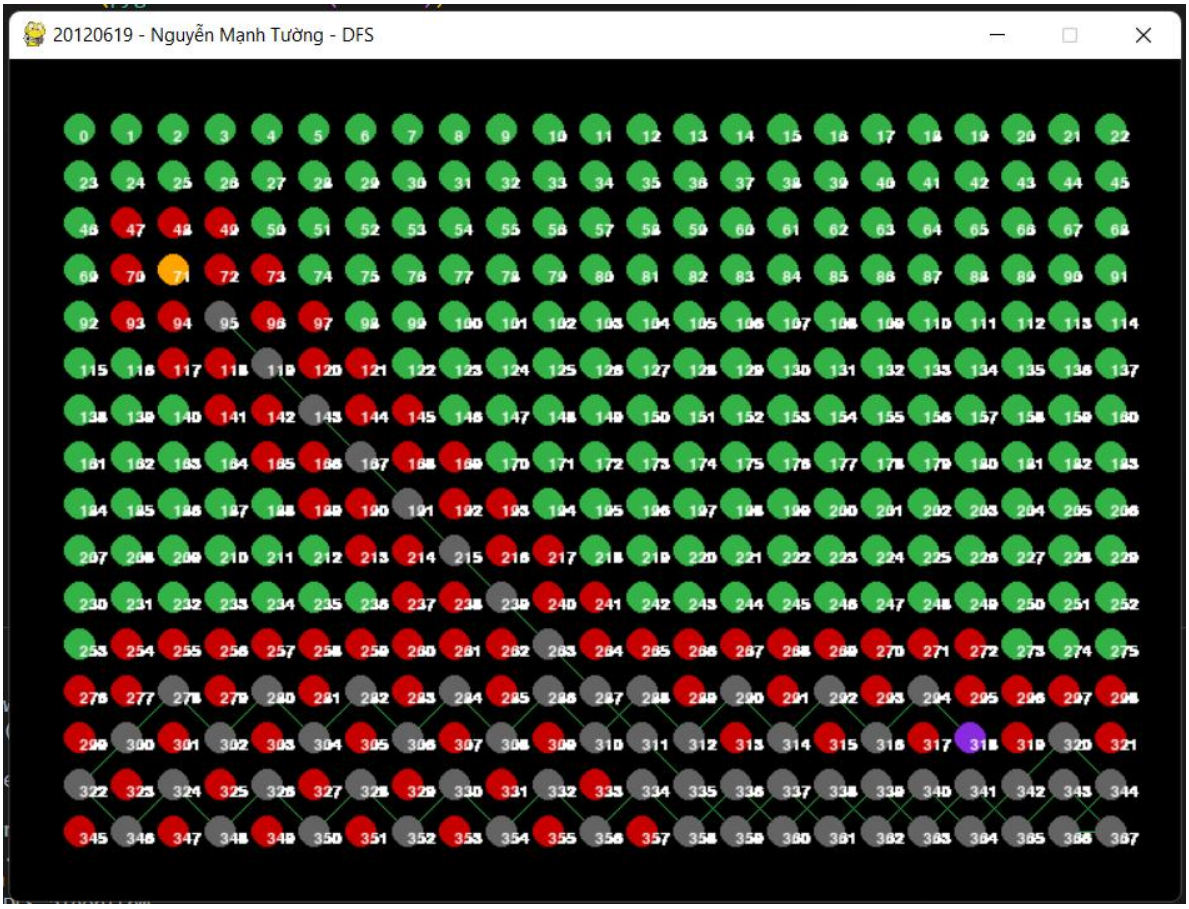
Bảng tóm tắt so sánh 2 thuật toán:

	Khởi tạo	Yêu cầu bộ nhớ	Thời gian chạy	Ứng dụng
Dijkstra	Môt tập hợp tất cả các đỉnh trong đồ thị	Cần lưu trữ toàn bộ đồ thị	Chậm hơn do yêu cần bộ nhớ lớn	Chỉ những đồ thị tường minh
UCS	Một tập hợp chỉ có các đỉnh nguồn	Chỉ lưu trữ các đỉnh cần thiết	Nhanh hơn do yêu cầu ít bộ nhớ hơn	Cả đồ thị tường minh và không tương minh

III. Mô tả các thuật toán đã code.

1. DFS

Kết quả đạt được:

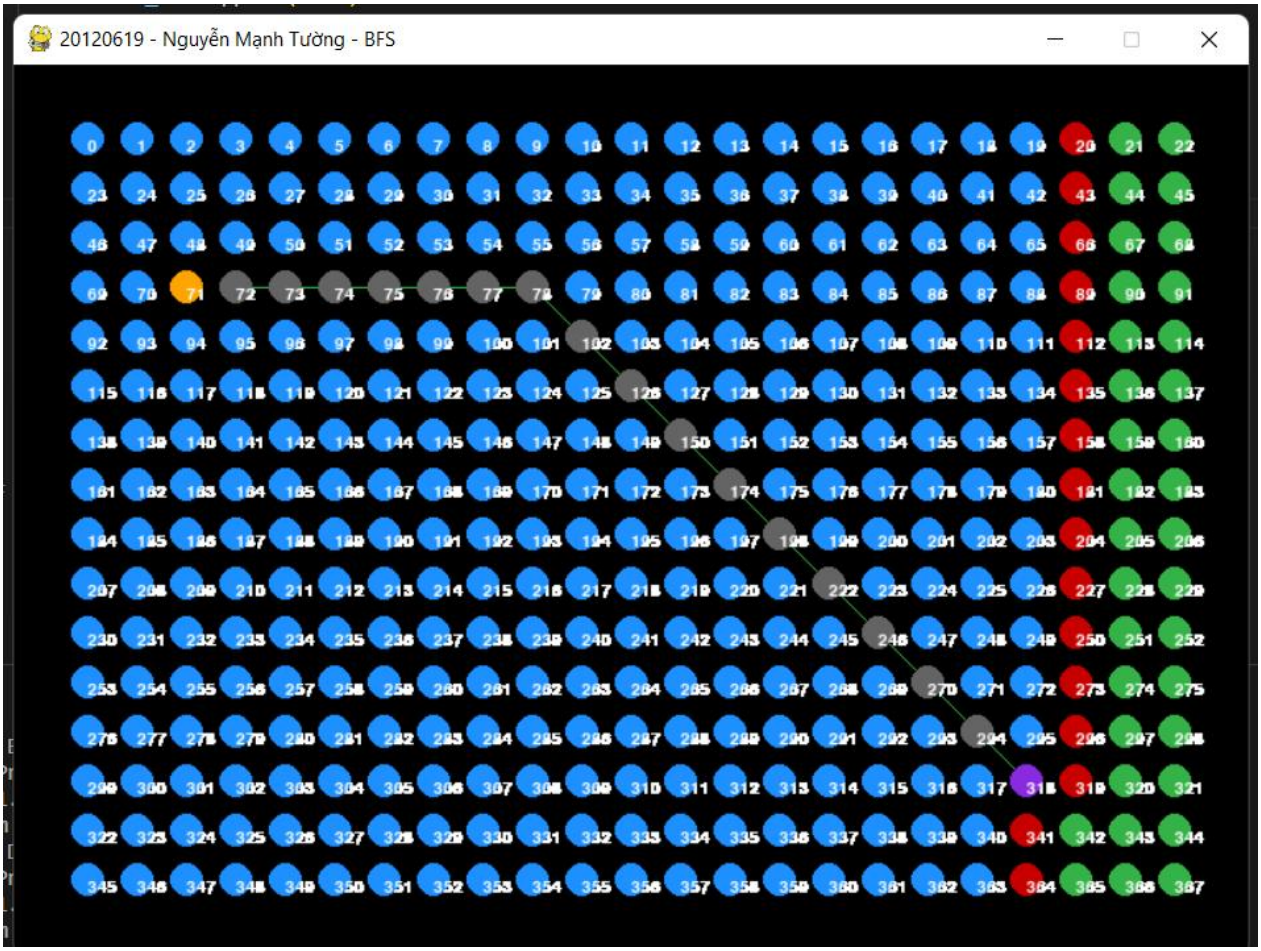


Mô tả: Quá trình tìm kiếm của thuật toán DFS. Duyệt các node theo chiều sâu cho đến khi tìm được đích nên đường đi dài và không tuân theo một cấu trúc nhất định.

Nhận xét về kết quả: Kết quả cho thấy việc tìm kết quả của DFS là không tối ưu. Mặc dù vẫn tìm ra kết quả. Về thời gian tìm kiếm lâu. Số lượng node cần lưu trữ nhiều.

2. BFS

Kết quả đạt được

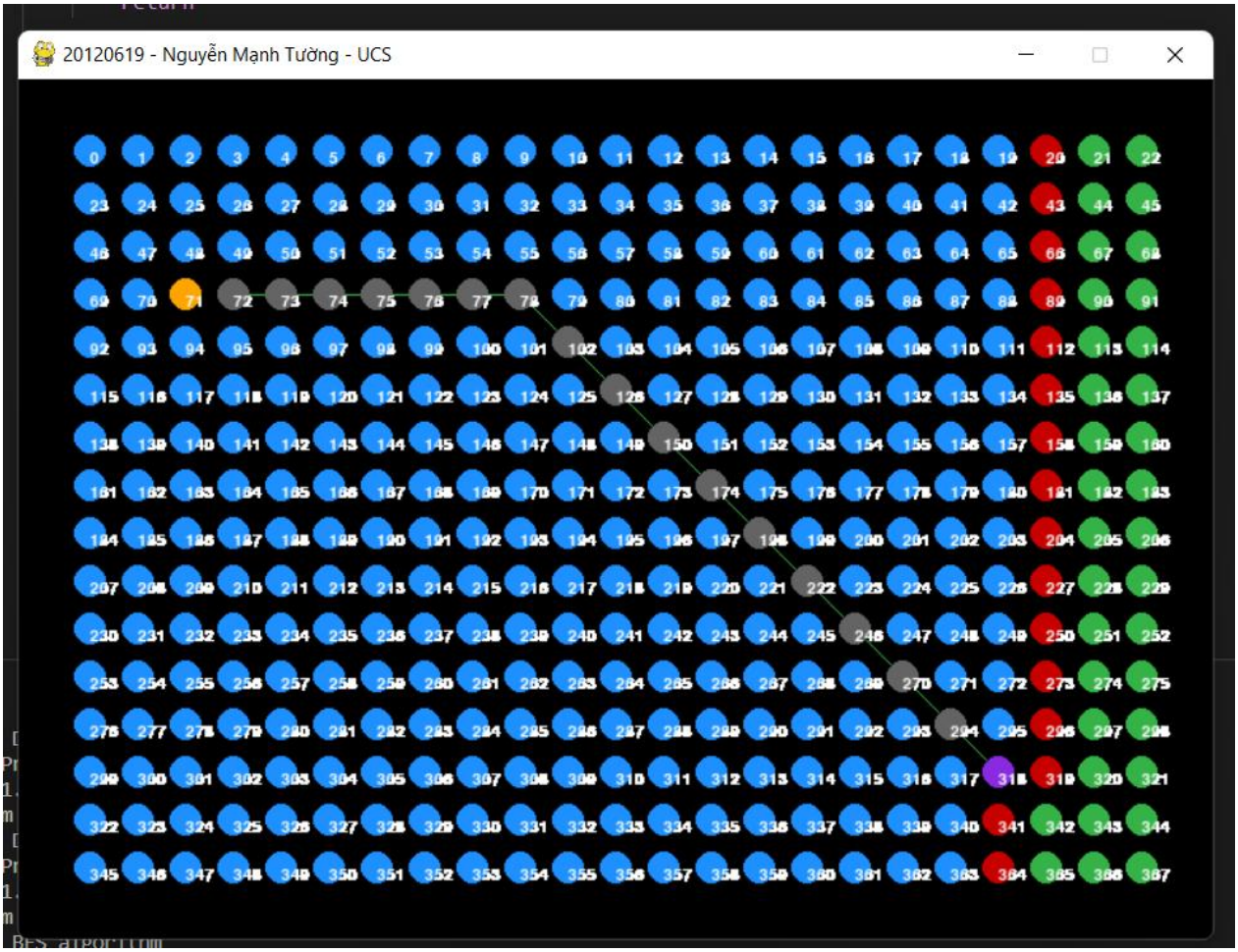


Mô tả: Vì thuật toán BFS duyệt qua tất các đỉnh kề nên nó sẽ tỏa ra như ngọn lửa đang cháy cho đến khi nào tìm được node đích.

Nhận xét về kết quả: Kết quả của tìm kiếm BFS cho ra kết quả tối ưu. Nhưng thời gian chạy khá lâu vì số lượng node cần lưu trữ lớn.

3. UCS

Kết quả đạt được:

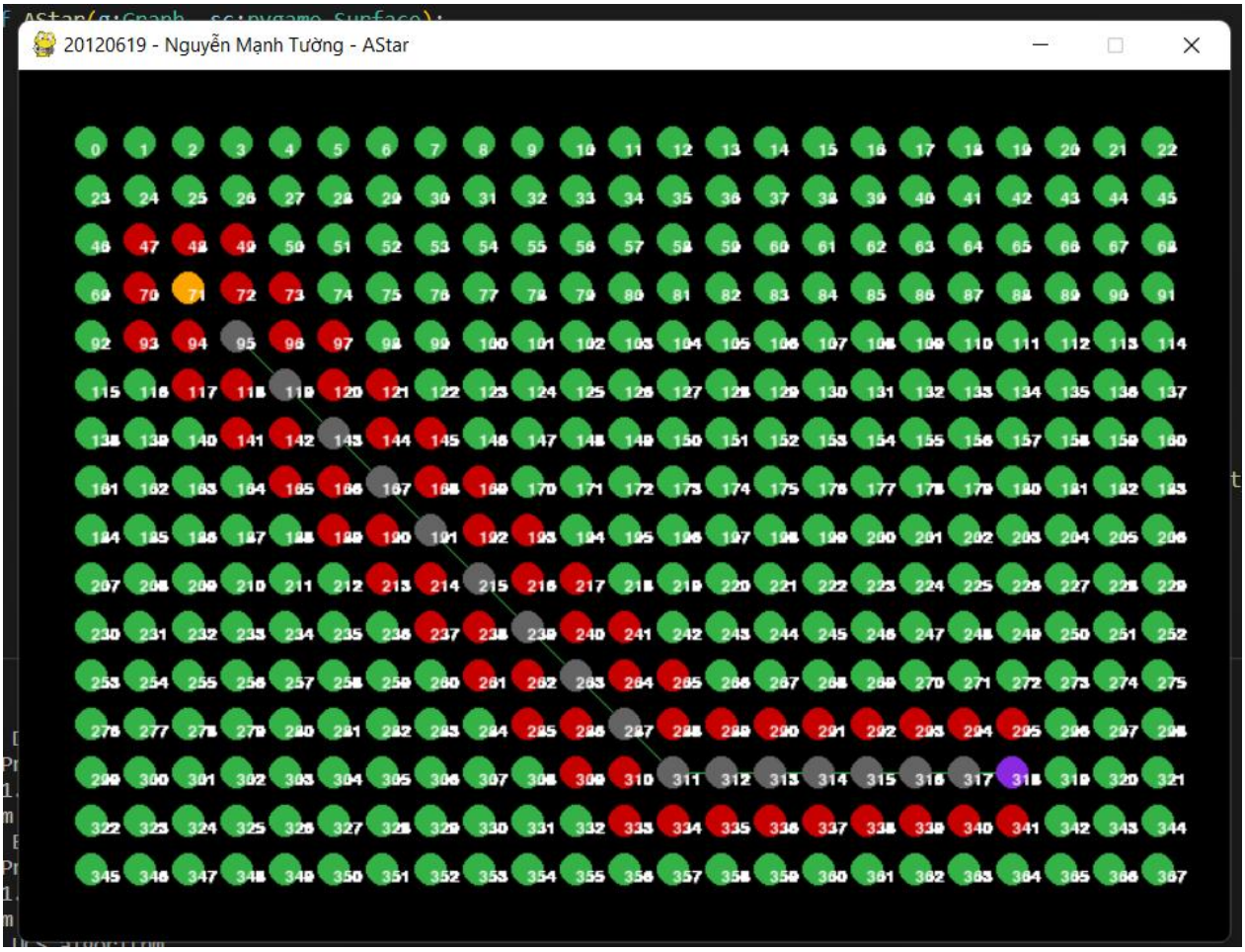


Mô tả: Vì khoảng cách giữa các node kề nhau đều bằng nhau nên quá trình tìm kiếm của thuật toán UCS khá giống với thuật toán BFS.

Nhận xét kết quả: Giống với BFS kết quả đạt được cho ra kết quả tối ưu. Nhưng thời gian chạy lâu vì số lượng node cần lưu trữ lớn

4. A*

Kết quả đạt được:



Mô tả: Vì hàm heuristic đảm bảo tính tối ưu nên quá trình tìm kiếm đến node đích đúng theo hướng tối ưu nhất.

Nhận xét:Vì hàm heuristic đảm tính tối ưu nên kết quả cuối cùng cho ra kết quả tối ưu. Thời gian tìm kiếm nhannh, không gian lưu trữ ít.

IV. Tìm hiểu thêm một thuật toán (Greedy)

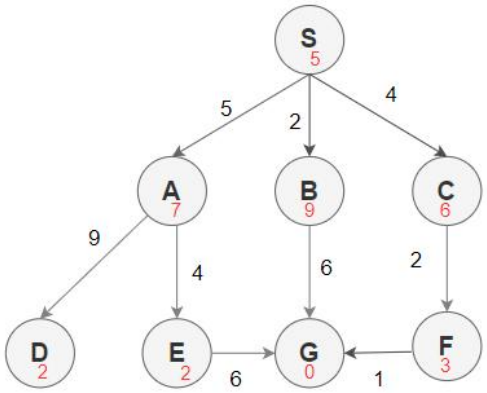
Ý tưởng: chọn trong tập nút biên nút có khoảng cách tới đích nhỏ nhất để mở rộng. Lý do làm như vậy để dễ mở rộng nút gần đích có xu hướng dẫn tới lời giải nhanh hơn. Trong phương pháp này, để đánh giá độ tốt của một nút ta dùng hàm heuristic. Đối với thuật toán Greedy thì f(n) = h(n).

Đánh giá thuật toán:

- Tính đầy đủ: Không có tính đầy đủ do có khả năng tạo thành vòng lặp vô hạn ở một số nút.
- Tính tối ưu: Thuật toán không tối ưu
- Độ phức tạp về thời gian và không gian: Trong trường hợp xấu nhất là O (b^m)

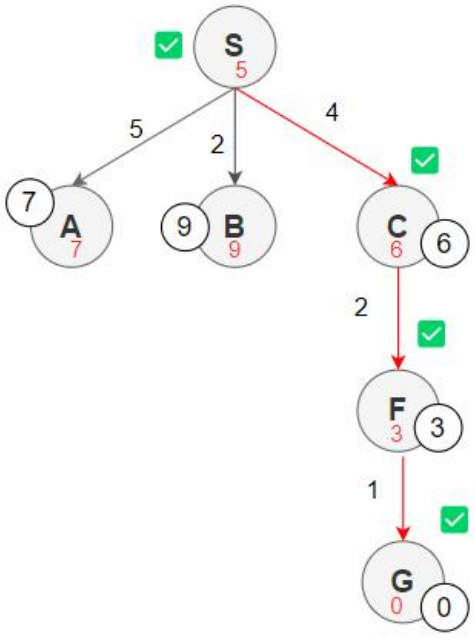
Ví dụ minh họa:

Đề bài sử dụng thuật toán Greedy tìm đường đi từ đỉnh S đến đỉnh G



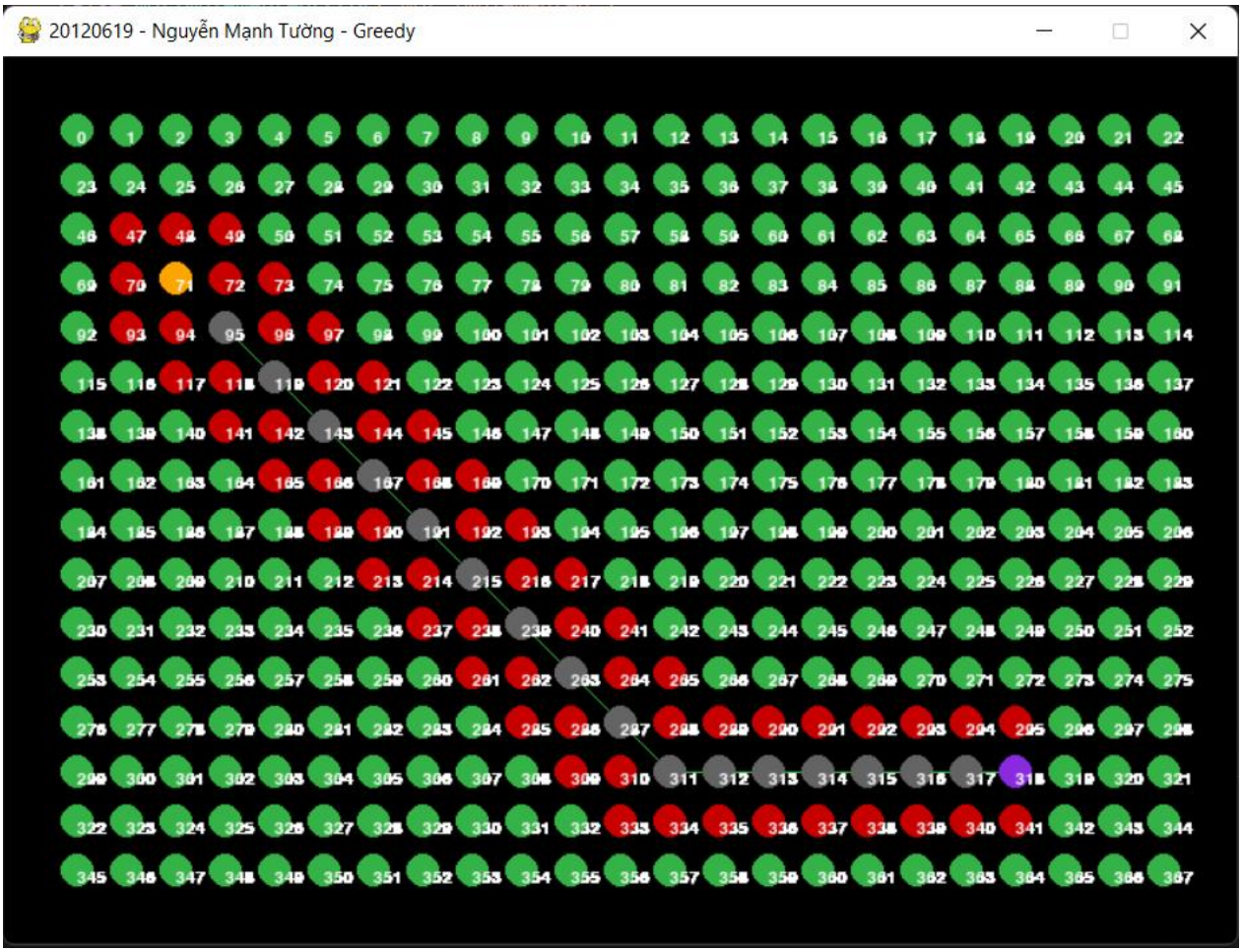
Bài làm:

Hình vẽ minh họa đường đi của thuật toán Greedy



Code tìm đường đi bằng thuật toán Greedy:

Kết quả đạt được:



Mô tả: Vì các đỉnh kề nhau có khoảng cách như nhau nên đường đi của thuật toán Greedy giống với thuật toán A*.

Nhận xét: Vì các đỉnh kề nhau có khoảng cách như nhau nên thuật toán Greedy cho ra đường đi tối ưu.

V Tham khảo và đánh giá

1. Tài liệu tham khảo:

https://fit.lqdtu.edu.vn/files/FileMonHoc/1.%20TTNT.pdf

https://www.pythonpool.com/a-star-algorithm-python/

https://drive.google.com/file/d/14LlgPi84-GZdl-fc-KUAU7BIPl-VYqj8/view

https://vnoi.info/wiki/algo/graph-theory/Depth-First-Search-Tree.md

https://vnoi.info/wiki/algo/graph-theory/breadth-first-search.md

https://viblo.asia/p/thuat-toan-tham-lam-greedy-algorithm-XQZGxozlvwA

https://stackoverflow.com/questions/12806452/whats-the-difference-between-uniform-cost-search-and-dijkstras-algorithm

https://www.geeksforgeeks.org/search-algorithms-in-ai/

2. Đánh giá

Yêu cầu	Đánh giá
Tìm hiểu và trình bày được các thuật toán tìm kiếm trên đồ thị	9.5 điểm
So sánh các thuật toán với nhau	9 điểm
Cài đặt được các thuật toán	9 điểm
Tìm hiểu thêm các thuật toán tìm kiếm ngoài.	7 điểm

Tổng trung bình của kết quả đánh giá: 9 điểm