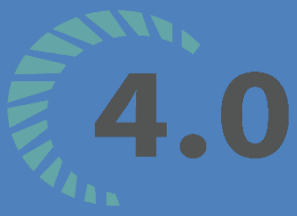
BỘ MÔN HỆ THỐNG THÔNG TIN – KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH, ĐẠI HỌC QUỐC GIA TP HCM

MÔN NHẬP MÔN TRÍ TUỆ NHÂN TẠO



Sinh viên thực hiện: 20120575 – Nguyễn Khắc Tấn

GV phụ trách: GS.TS. Lê Hoài Bắc

BÀI TẬP MÔN HỌC - MÔN NHẬP MÔN TRÍ TUỆ NHÂN TẠO

HỌC KỲ II – NĂM HỌC 2022 – 2023

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện Thoại: (08) 38.354.266 - Fax:(08) 38.350.096

YÊU CẦU BÀI TẬP

Mục lục

A. Yêu cầu của Bài tập

Thực hiện các yêu cầu bài tập theo file Requirement.pdf

B. Kết quả

1. Các thành phần thuật toán tìm kiếm. Phân biệt

1.1. Các thành phần thuật toán tìm kiếm

- Tập dữ liệu đầu vào: đây là tập hợp các phần tử mà thuật toán tìm kiếm sẽ xử lý
- Mục tiêu tìm kiếm: đây là giá trị hoặc điều kiện mà thuật toán tìm kiếm sẽ tìm trong tập dữ liệu đầu vào
- Hàm đánh giá: đây là hàm sử dụng để đánh giá xem một phần tử có phù hợp với mục tiêu tìm kiếm hay không
- Phương pháp tìm kiếm: đây là phương pháp được sử dụng để tìm kiếm mục tiêu trong tập dữ liệu đầu vào, ví dụ như tìm kiếm tuyến tính, tím kiếm nhị phân, tìm kiến theo đường đi

- Cấu trúc dữ liệu: đây là cách mà dữ liệu được tổ chức và lưu trữ trong bộ nhớ, và nó có thể ảnh hưởng đến hiệu quả của thuật toán tìm kiếm
- Các thao tác: đây là các hoạt động được thực hiện trên các phần tử trong tập dữ liệu đầu vào

1.2. Phân biệt Informed search và Uniformed search

Informed search	Uniformed search
Sử dụng các tri thức cụ thể của bài toán, quá trình tìm kiếm hiệu quả hơn	Chỉ sử dụng các thông tin chứa trong định nghĩa
Phù hợp với các bài toán thực tế hơn	Đòi hỏi chi phí quá cao về thời gian và bộ nhớ, nên không phù hợp với nhiều bài toán trong thực tế
Bao gồm các thuật giải: - Heuristic - Tìm kiếm tham lam (Greedy Search) - Thuật giải A*	Bao gồm các thuật toán: - Tìm kiếm theo chiều sâu (DFS) - Tìm kiếm theo chiều rộng (DFS) - Tìm kiếm với chi phí đồng nhất (UCS0 - Tìm kiếm với lặp sâu dần (Iterative Deepening)

2. Cách giải (bằng mã giả) của bài toán tìm kiếm

Input:

Problem: thông tin bài toán

Strately: chiến lược duyệt

Output:

Solution: nếu tồn tại lời giải

Failure: nếu không tồn tại lời giải

Khởi tạo cây tìm kiếm với trạng thái bắt đầu của Problem

Loop

If không còn ứng viên mở rộng then return failture

Chọn một node lá để mở rộng dựa theo Strately

If node được chọn là trạng thái đích then return Solution

Else mở rộng node và thêm các node kết quả vào cây tìm kiếm

End

3. Các thuật toán tìm kiếm

3.1. Thuật toán tìm kiếm DFS

3.1.1. Ý tưởng chung: mở rộng node sâu nhất trước

Là thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị và sử dụng ngăn xếp (stack) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào. Thuật toán tiếp tục cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó thuật toán quay lui về đỉnh vừa mới tìm kiếm ở bước trước

3.1.2. Mã giả

```
Đặt P = <START (expand = succs(START))>
While (P khác rỗng và top(P) không là đích)
        if mở rộng của top(P) rỗng
        then
                loại bỏ top(P) (“pop ngăn xếp”)
        else
                gọi s một thành viên của mở rộng của top(P)
                loại s khỏi mở rộng của top(P)
                tạo một mục mới trên đỉnh đường đi P:
                        s (expand = succs(s))
If P rỗng
        trả về FAILURE
Else
        trả về đường đi chứa trạng thái của P
```

Thuật toán này có thể được viết gọn dưới dạng đệ qui, dùng ngăn xếp của chương trình để cài đặt P.

3.1.3. Các thuộc tính của thuật toán (completeness, optimal, time complexity, space complexity)

Với không gian tìm kiếm không có chu trình:

a. Completeness: có, với không gian tìm kiếm không có chu trình luôn đảm tìm được đường đi

b. Optimal

Không, vì luôn tìm thấy lời giải “trái nhất”, bất chấp độ sâu hay chi phí, thuật toán DFS không đảm bảo tìm được giải pháp tối ưu.

c. Time complexity

Với m là độ dài đường đi dài nhất từ start đến bất cứ đâu và B là thừa số phân nhánh trung bình, thì độ phức tạp thời gian là: O(B m)

d. Space complexity

Với m là độ dài đường đi dài nhất từ start đến bất cứ đâu, thì độ phức tạp không gian: O(m)

Với không gian tìm kiếm có chu trình:

a. Completeness: không

b. Optimal: không

c. Time complexity: N/A

d. Space complexity: N/A

3.2. Thuật toán tìm kiếm BFS

3.2.1. Ý tưởng chung: chiến lược duyệt: mở rộng node ở mức cạn nhất trước

Thuật toán BFS (Breadth-first search) từ một gốc sẽ duyệt qua tất cả đỉnh con liền kế (nút con trên cây của nó). Lưu các đỉnh con kề vào stack, sau đó từ các đỉnh con này ta lại tiếp tục thực hiện tìm kiếm tất cả các đỉnh con của các đỉnh con đó. Cứ thực hiện lặp như vậy cho tới khi nào không còn một nút nào trong cây hoặc đồ thị hay tìm được đỉnh đích, khi đó thuật toán kết thúc

3.2.2. Mã giả

V_0 := S (tập các trạng thái ban đầu)
previous(*START*) := *NIL*
k := *0*
while (không có trạng thái đích trong V_k và V_k khác rỗng) **do**
    V_{k+1} := tập rỗng
    Với mỗi trạng thái s trong V_k
        Với mỗi trạng thái s' trong **succs**(s)
            Nếu s' chưa gán nhãn
                Đặt *previous*(s') := s
                Thêm s' vào V_{k+1}
    k := k+*1*
If V_k rỗng thì FAILURE
Else xây dựng lời giải: Đặt S_i là trạng thái thứ i trên đường đi ngắn nhất. Định nghĩa S_k = GOAL, và với mọi i <= k, định nghĩa S_{i-1} = *previous*(S_i).

3.2.3. Các thuộc tính của thuật toán (completeness, optimal, time complexity, space complexity)

Với B là thừa số phân nhánh trung bình (số con trung bình) (B>1)

L là độ dài đường đi từ start tới goal với số bước (chi phí) ít nhất

a. Completeness: có, vì L là xác định nếu tồn tại lời giải

b. Optimal: có, chi khi các chi phí đều bằng nhau

c. Time complexity: O(min(N,BL))

d. Space complexity: O(min(N,BL))

3.3. Thuật toán tìm kiếm UCS

3.3.1. Ý tưởng chung: chiến lược mở rộng: mở rộng node có chi phí rẻ nhất trước (là chi phí từ trạng thái bắt đầu đến node đó)

Ý tưởng của thuật toán UCS là tìm kiếm theo chiều rộng nhưng thay vì sử dụng một hàng đợi FIFO (First-In-First_out) để duyệt các đỉnh, ta sử dụng một hàng đợi ưu tiên (priority queue) để xắp các đỉnh theo chi phí đường đi hiện tại đến đỉnh đó

Thuật toán bắt đầu bằng cách đưa đỉnh bắt đầu vào hàng đợi ưu tiên với chi phí đường đi hiện tại bằng 0. Sau đó lấy đỉnh đầu tiên trong hàng đợi, kiểm tra nó phải đỉnh kết thúc không. Nếu không phải thì duyệt qua các đỉnh kề với đỉnh hiện tại và thêm vào hàng đợi ưu tiên với chi phí đường đi là chi phí đường đi hiện tại cộng với chi phí từ đỉnh hiện tại đến đỉnh kế tiếp

Trong quá trình tìm kiếm, thuật toán UCS duyệt các đỉnh theo thứ tự tăng dần của chi phí đường đi hiện tại đến đỉnh đó, đảm bảo rằng tìm được đường đi ngắn nhất có thể

3.3.2. Mã giả

V_k = tập các trạng thái có thể đến được trong đúng k bước, và với nó đường đi k-bước chi phí thấp nhất thì ít chi phí hơn bất kỳ đường đi nào có độ dài nhỏ hơn k. Nói cách khác, V_k = tập trạng thái mà giá trị của nó thay đổi so với vòng lặp trước.

V_0 := S (tập trạng thái đầu)
previous(*START*) := *NIL*
$g(START) = 0$
$k := 0$
while (V_k khác rỗng) **do**
    V_{k+1} := rỗng
    Với mỗi s trong V_k
        Với mỗi s' trong **succs**(s)
            Nếu s' chưa được gán nhãn
            HAY nếu $g(s) + Cost(s,s') < g(s')$
                Đặt *previous*(s') := s
                Đặt $g(s') := g(s) + Cost(s,s')$
                Thêm s' vào V_{k+1}
    $k := k+1$
Nếu GOAL chưa gán nhãn, thoát FAILURE
Ngược lại xây dựng lời giải theo: Đặt S_k là trạng thái thứ k trên đường đi ngắn nhất. Định nghĩa S_k = GOAL, và với mọi $i <= k$, định nghĩa S_{i-1} = *previous*(S_i).

3.3.3. Các thuộc tính của thuật toán (completeness, optimal, time complexity, space complexity)

Với:

- N là số trạng thái trong bài toán
- B là thừa số phân nhánh trung bình (số con trung bình) (B>1)
- L là độ dài đường đi từ start đến goal với số bước (chi phí) nhỏ nhất
- Q là kích cỡ hàng đợi trung bình

a. Completeness: có, nếu lời giải "nhỏ nhất" có chi phí xác định và tất cả các cạnh có chi phí không âm

b. Optimal: có, đảm bảo tìm được đường đi ngắn nhất từ đỉnh bắt đầu đến đỉnh kết thúc

c. Time complexity: O(log(Q) * min(N,BL))

d. Space complexity: O(min(N,BL))

*3.4. Thuật giải tìm kiếm A**

3.4.1. *Ý tưởng chung:* kết hợp UCS và tìm kiếm tham lam

Là thuật giải tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic

Từ trạng thái hiện tại A* xây dựng tất cả các đường đi có thể đi dùng hàm ước lược khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế

Sử dụng hàm đánh giá f(n) = g(n) + h(n)

- g(n) = chi phí nút gốc cho đến nút hiện tại
- h(n) = chi phí ước lượng từ nút hiện tại n tới đích
- f(n) = chi phí tổng thể ước lượng của đường đi qua nút hiện tại đến đích

3.4.2. Mã giả

1. Khởi tạo tập **OPEN** = {start}, **CLOSED** = {}, g(start) = 0, f(start) = h(start)
2. If OPEN == {} then return failure
3. Chọn trạng thái có chi phí nhỏ nhất từ **OPEN**, n. Đưa n vào **CLOSED**.
4. If n là trạng thái đích then return solution
5. Mở rộng node n. Với mỗi m là node con của n, ta có:

    If m không thuộc **OPEN** hay **CLOSED** then:

        Gán g(m) = g(n) + C(n, m)

        Gán f(m) = g(m) + h(m)

        Thêm m vào **OPEN**

    If m thuộc **OPEN** hoặc **CLOSED** then:

        Gán g(m) = min {g(m), g(n) + cost(n, m)}

        Gán f(m) = g(m) + h(m)

    If f(m) giảm và m thuộc **CLOSED** then:

        Đưa m trở lại **OPEN**

End

Quay lại bước 2.

3.4.3. Các thuộc tính của thuật toán (completeness, optimal, time complexity, space complexity)

a. Completeness

Nếu không gian các trạng thái là hữu hạn và có giải pháp để tránh việc xét (lặp) lại các trạng thái, thì thuật giải A* là hoàn chỉnh, ngược lại thì thuật giải A* không hoàn chỉnh

b. Optimal

Chúng ta cần các ước lượng nhỏ hơn chi phí thực tế

Một ước lượng h(n) được xem là chấp nhận được nếu đối với mọi nút n: 0 ≤ h(n) ≤ h * (n), trong đó h * (n) là chi phí thật (thực tế) để đi từ nút n đến đích, lúc đó thuật giải A* là tối ưu

c. Time complexity

Bậc của hàm mũ – số lượng các nút được xét là hàm mũ của độ dài đường đi của lời giải

d. Space complexity: lưu trữ tất cả các nút trong bộ nhớ

3.4.4. Định nghĩa Heuristic trong A. Liệt kê một số phương án của Heuristic*

a. Tạo các Heurustic có thể chấp nhận

Một ước lượng h(n) được xem là chấp nhận được nếu đối với mọi nút n:

$$0 \le h(n) \le h^*(n)$$

Trong đó h * (n) là chi phí thật (thực tế) để đi từ nút n đến đích.

Thông thường, Heuristic chấp nhận được là các lời giải cho các bài toán được nới lỏng ràng buộc (relaxation), ở đó có nhiều hành động mới được cho phép

Đôi khi các Heuristic không chấp nhận được cũng có ích

b. Sự nới lỏng (relaxation)

Để có một Heuristic tốt thì h(s) = h*(s), nhưng điều này khó giải quyết bài toán gốc, thay vì dùng h*(s) của bài toán gốc, chúng ta dùng h*(s) của bài toán dễ hơn

Heuristic tốt liên quan đến mô hình hóa, không phải xây dựng thuật toán

4. So sánh

*4.1. So sánh UCS, Greedy và A**

UCS, Greedy	A*
Tìm kiếm với chi phí cực tiểu (UCS), tham lam (Greedy) phát triển theo mọi hướng	Tìm kiếm A* phát triển chủ yếu theo hướng tới đích, nhưng vẫn đảm bảo tính tối ưu

4.2. So sánh UCS và Dijkstra

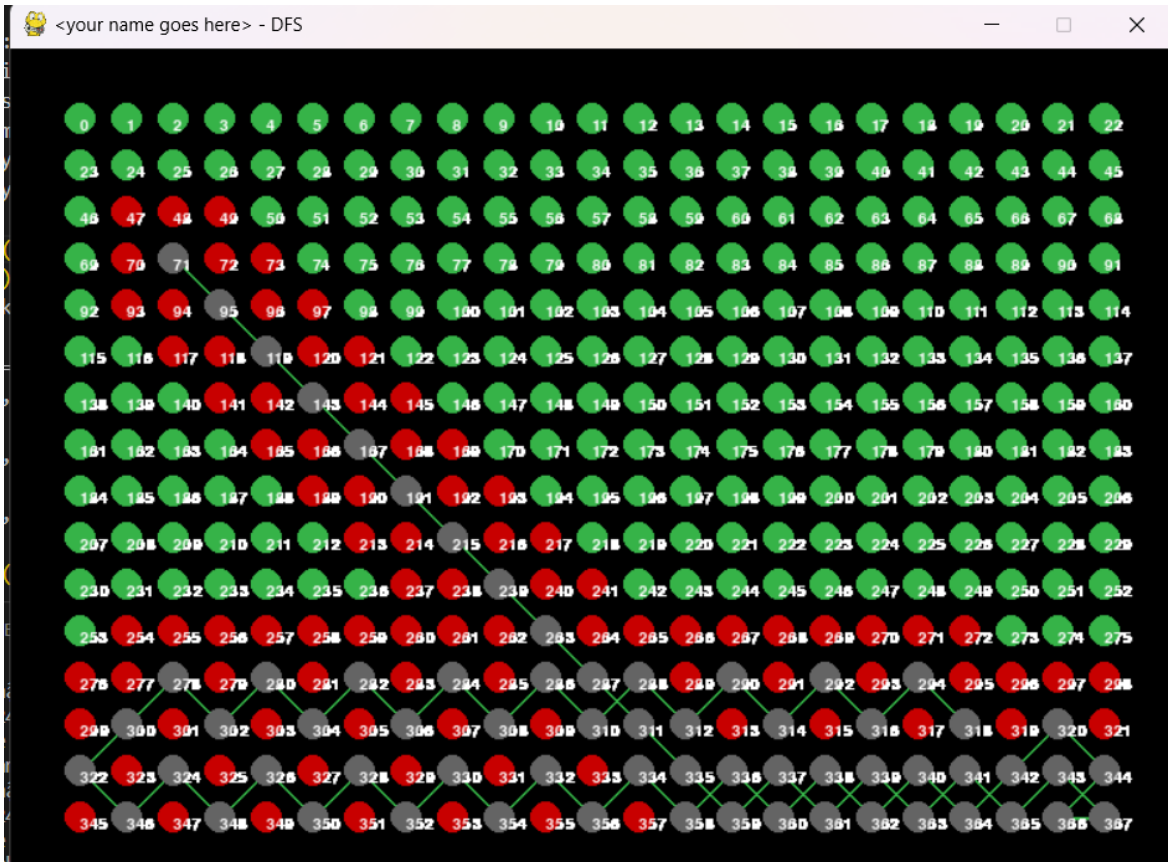
UCS	Dijkstra

Tìm kiếm đường đi ngắn nhất từ nút bán đầu đến nút mục tiêu, thay vì con đường ngắn nhất đến mọi nút.	Tìm kiếm các đường đi ngắn nhất từ gốc đến mọi nút khác trong biểu đồ
Yêu cầu không gian ít hơn	Yêu cầu độ phức tạp không gian lớn, do thêm tất cả các nút vào hàng đợi
Kết thúc khi tìm thấy nút mục tiêu	Kết thúc khi tất cả các nút bị xóa khỏi hàng đợi ưu tiên
Thường được xây dựng trên cây Bắt đầu với đỉnh đầu và dần dần đi qua có phần cần thiết, do đó nó có thể áp dụng cho cả đồ thị rõ ràng và đồ thị ẩn	Thướng được sử dụng trên đồ thị chung, chỉ được áp dụng trên trong các đồ thị rõ ràng, nơi biết tất cả các đỉnh và cạnh

5. Thực hành

Youtube URL: [Search in graph] - 20120575 - HCMUS - YouTube

Thuật giải tìm kiếm DFS:

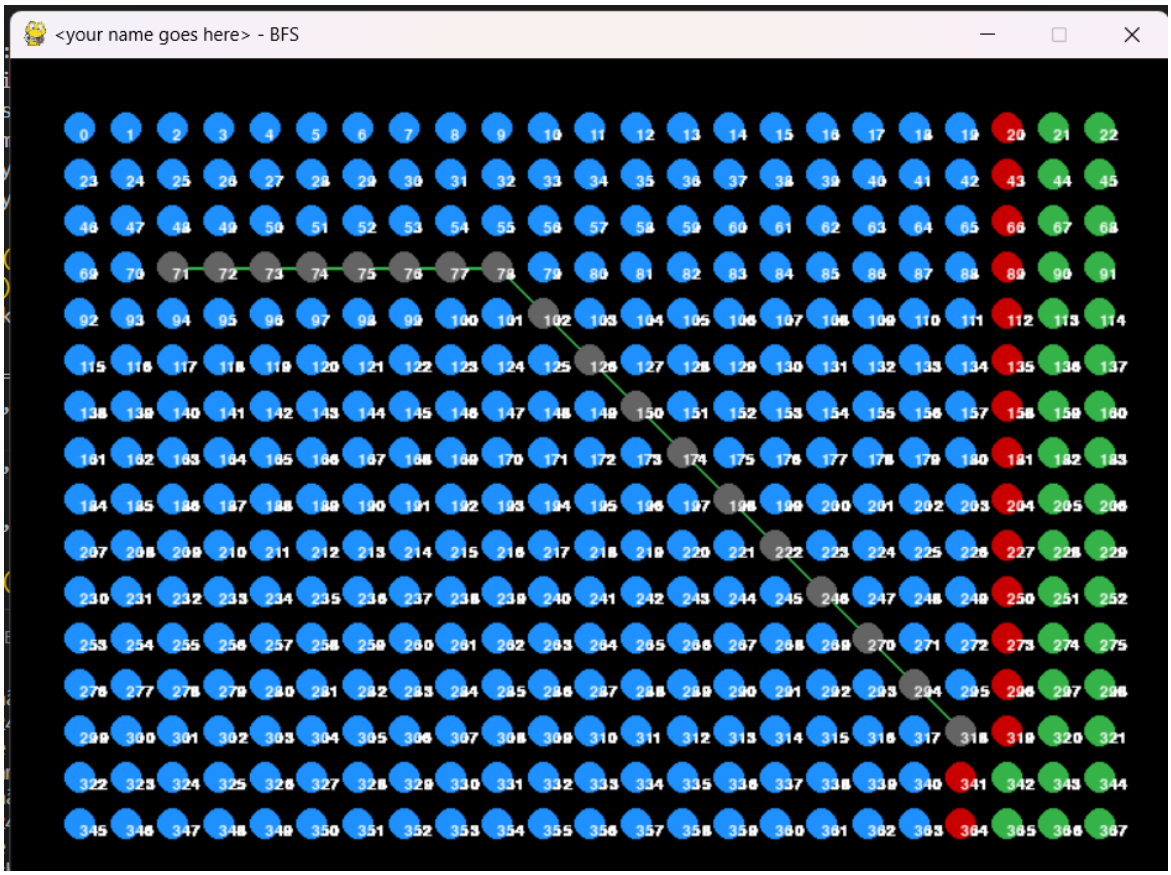


Mô tả thuật giải DFS:

- Khởi tạo open_set, closed_set, father
- Trong khi open_set khác rỗng:

+ current_node_value = open_set.pop()
+ Thêm current_node_value vào closed_set
+ Kiểm tra current_node_value có phải là node đích không, nếu phải thì in ra đường đi, kết thúc
+ Đổi màu current_node_value sang màu vàng
+ Lấy có giá trị node con của current_node_value
+ Duyệt từng node con:
    Nếu node con chưa xuất hiện trong closed_set:
        Thêm node con vào open_set, father
        Đổi node con sang màu đỏ
+ Đổi màu current_node_value sang màu xanh

Thuật giải tìm kiếm BFS:

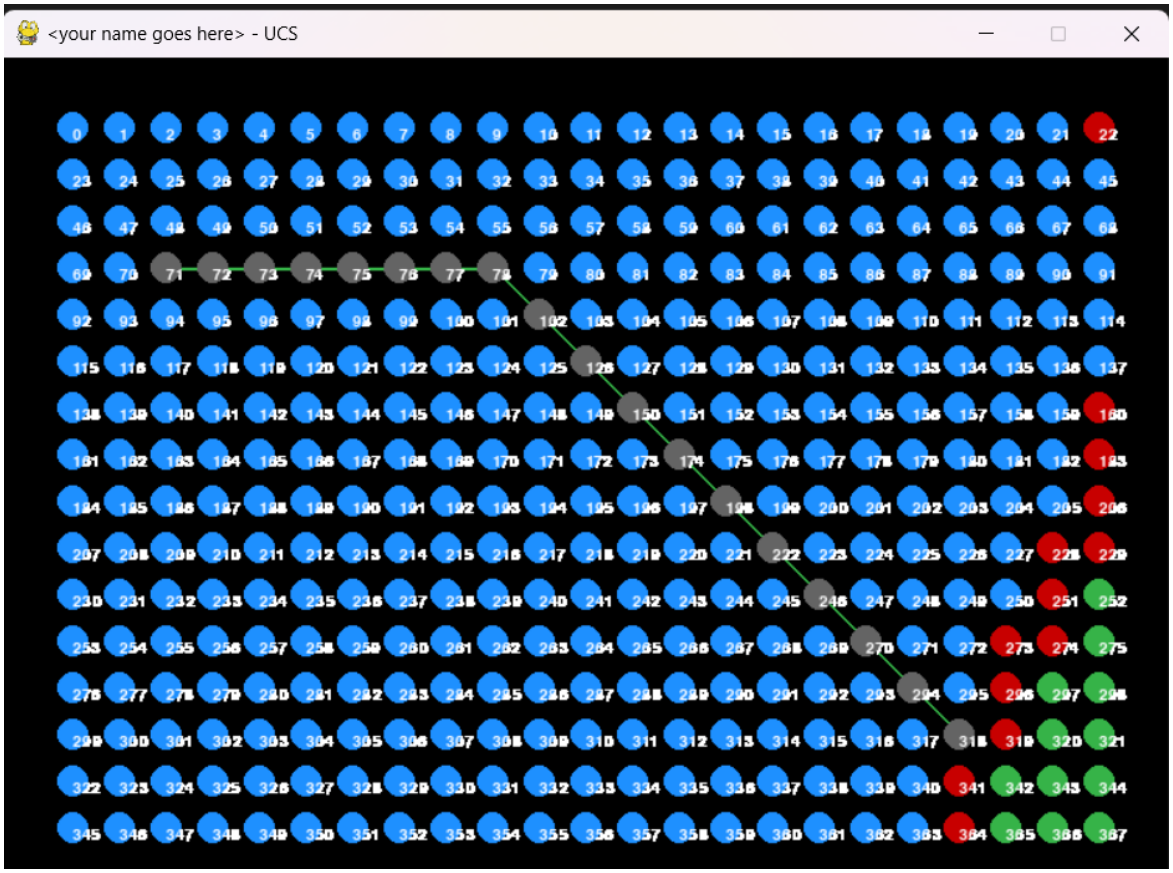


Mô tả thuật giải tìm kiếm BFS:

- Khởi tạo open_set, closed_set, father
- Trong khi open_set khác rỗng:
    + Lấy node: current = open_set()
    + Đổi màu node sang màu vàng
    + Nếu node hiện tại là node đích:
        In ra đường đi, kết thúc

+ Duyệt từng node con của current (node hiện tại)
    Nếu node con không xuất hiện trong closed_set hoặc open_set
        Thêm vào opent_set, father
        Đổi màu node con sang màu đỏ
+ Đổi màu current sang màu xanh

Thuật giải tìm kiếm UCS:



Mô tả thuật giải tìm kiếm UCS:

- Khởi tạo open_set, closed_set, father, cost
- Trong khi open_set khác rỗng:
  + Lấy node: current_node = min(open_set, key = lambda k: open_set[k])
  + current_cost = open_set[current_node]
  + Xóa current_node trong open_set, thêm current_node vào closed_node
  + Đổi màu current_node sang màu vàng
  + Kiểm tra current_node có phải g.goal không:
      Nếu đúng, in đường đi, kết thúc
  + Lấy tọa độ của current_node
  + Duyệt các node con của current_node
      +Nếu node con không xuất hiệnh trong closed_set
          +Lấy tọa độ của node_con

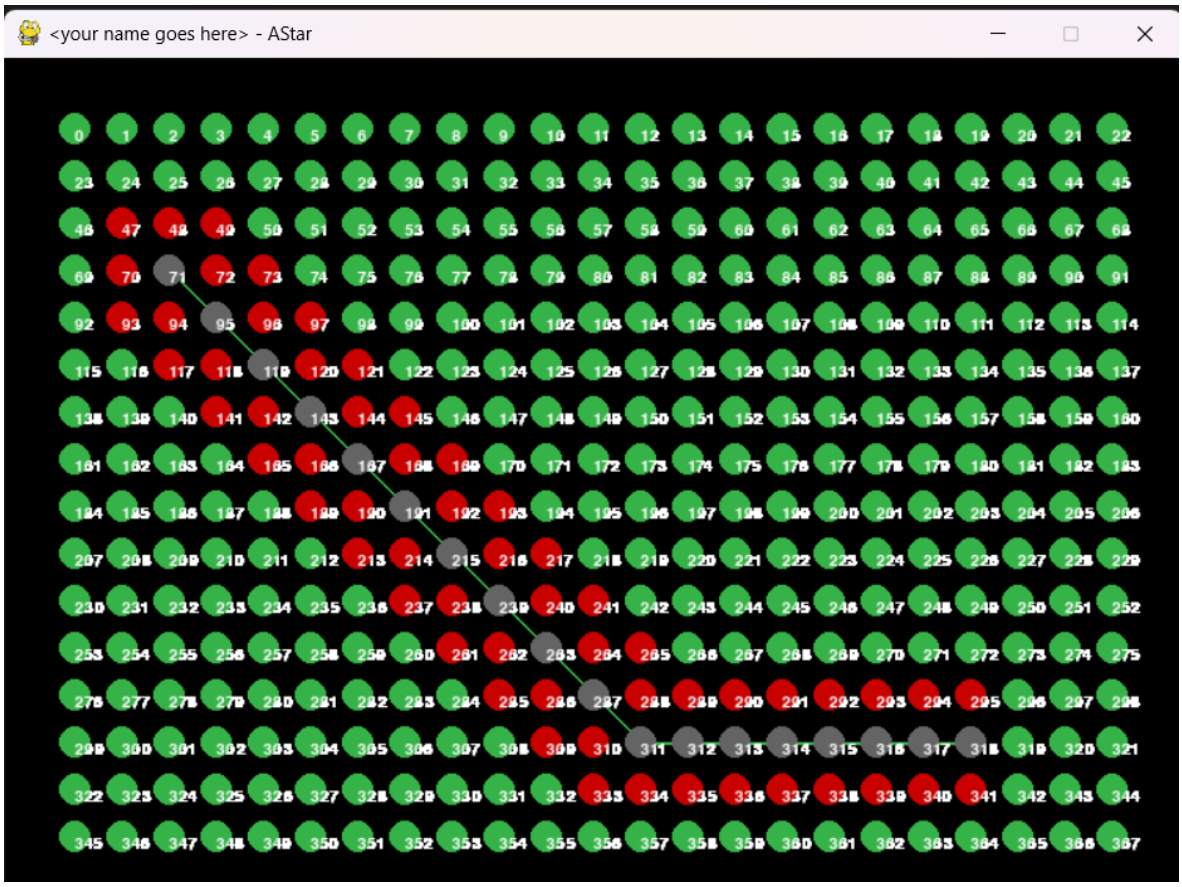
+Tính chi phí từ tọa độ 2 node và chi phí từ node đầu đến node hiện tại, so sánh với chi phí của node con

+Nếu nhỏ hơn:

+ cost[node con] = chi phí nhỏ hơn
+ thêm node con vào father
+ Xét màu node con sang màu đỏ

+ Đổi màu current_node sang màu xanh

Thuật giải tìm kiếm A*:



Mô tả thuật giải A*:

- Khởi tạo open_set, closed_set, father, cost
- Trong khi open_set khác rỗng:
  - Lấy node: current_node_value = mion(open_set, key=open_set.get)
  - Xóa current_node_value khỏi open_set
  - Thêm current_node_value vào closed_set
  - Đổi màu current_node_value sang màu vàng
  - Kiểm tra current_node_value có phải g.goal không
    - Nếu đúng, in đường đi và kết thúc

- Duyệt các node con của current_node_value:
  - Nếu node con không có trong closed_set:
    - Tính toán chi phí tentative_f_cost = cost[current_node_value] + heuristic[node con, g.goal]
    - Nếu chi phí nhỏ hơn cost[nodecon], hoặc node chưa xuất hiện trong open_set:
      - Thêm node con vào father
      - Gán lại cost[node con]
      - Đổi màu node con sang màu đỏ
- Đổi màu current_node_value sang màu xanh