BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KỲ

Đề tài: THUẬT TOÁN A* VÀ MỘT SỐ ỨNG DỤNG

Giảng viên hướng dẫn : Trần Tiến Đức
Sinh viên thực hiện : Nguyễn Thị Hồng Thơ
MSSV : 22151305
Lớp : 22133B
Khóa : 2022
Mã lớp : ARIN330585_23_2_05

Thành phố Hồ Chí Minh, tháng 05 năm 2024

DANH SÁCH THAM GIA ĐỀ TÀI

HỌC KÌ I, NĂM HỌC: 2023 – 2024
Lớp: ARIN330585_23_2_05

Tên đề tài: Thuật toán A* và một số ứng dụng

HỌ VÀ TÊN SINH VIÊN	MÃ SỐ SINH VIÊN
Nguyễn Thị Hồng Thơ	22151305

Nhận xét của giảng viên

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Ngày 19 tháng 05 năm 2024

Giảng viên chấm điểm

DANH MỤC HÌNH ẢNH

MỤC LỤC

PHẦN MỞ ĐẦU

1. Lý do chọn đề tài

Trí tuệ nhân tạo là một lĩnh vực của khoa học máy tính với mục tiêu nghiên cứu xây dựng và ứng dụng các hệ thống thông minh nhân tạo. Đây là một trong những lĩnh vực được quan tâm nghiên cứu nhiều nhất của khoa học máy tính hiện nay với nhiều kết quả ứng dụng rộng rãi. Và khi quan sát các bài toán trên thực tế, có thể nhận thấy một lớp lớn bài toán có thể phát biểu và giải quyết dưới dạng tìm kiếm, trong đó yêu cầu có thể là tìm những trạng thái, tính chất thỏa mãn một số điều kiện nào đó, hoặc tìm chuỗi hành động cho phép đạt tới trạng thái mong muốn. Một số bài toán ứng dụng vào thực tế hiện nay như:

- Trò chơi: nhiều trò chơi, ví dụ cờ vua, thực chất là quá trình tìm kiếm nước đi của các bên trong số những nước mà luật chơi cho phép.

- Tìm đường đi: cần tìm đường đi từ điểm xuất phát tới đích, có thể thỏa mãn thêm một số tiêu chí nào đó như tiêu chí tối ưu về độ dài, thời gian, giá thành v.v.

Không những vậy, tìm kiếm còn là cơ sở cho rất nhiều nhánh nghiên cứu khác của trí tuệ nhân tạo như lập kế hoạch, học máy, xử lý ngôn ngữ tự nhiên, suy diễn. Sự phổ biến của các vấn đề có tính chất tìm kiếm dẫn tới yêu cầu phát biểu bài toán tìm kiếm một cách tổng quát, đồng thời xây dựng phương pháp giải bài toán tìm kiếm sao cho hiệu quả, thuận lợi. Và một trong những phương pháp tìm kiếm được xem là có hiệu quả, tổng quát là thuật toán tìm kiếm A* - một trong những dạng tìm kiếm có thông tin (Informed search) hay còn được gọi là tìm kiếm Heuristic sử dụng thêm thông tin từ bài toán để định hướng tìm kiếm giúp giải quyết bài toán hiệu quả hơn.

Nhận thấy những ứng dụng thực tế vào đời sống của thuật toán tìm kiếm nói chung và thuật toán A* nói riêng, em đã chọn đề tài: **"Thuật toán A* và một số ứng dụng"** để nghiên cứu, tìm hiểu rõ hơn về thuật toán.

2. Kết cấu đề tài

Ngoài phần Mở đầu, Kết luận, Danh mục tài liệu tham khảo, nội dung của đề tài được kết cấu thành 2 chương:

Chương 1: Tổng quan về tìm kiếm và thuật toán A*

Chương 2: Một số ứng dụng của thuật toán A*

CHƯƠNG 1: TỔNG QUAN VỀ TÌM KIẾM VÀ THUẬT TOÁN A*

1. Giải quyết vấn đề bằng tìm kiếm

Trong một không gian trạng thái của một bài toán, ta có thể xây dựng một cấu trúc đồ thị với các nút là các trạng thái của bài toán, các cung là phép chuyển trạng thái. Đồ thị này gọi là không gian trạng thái của bài toán.

Tìm kiếm trong không gian trạng thái là phương pháp được dùng phổ biến trong một lớp lớn các bài toán và lớp kỹ thuật để giải quyết những vấn đề quan trọng, được nghiên cứu và ứng dụng nhiều trong trí tuệ nhân tạo.

Các giải thuật tìm kiếm được chia thành ba nhóm lớn:

1. Tìm kiếm mù (không có thông tin)
2. Tìm kiếm heuristics (có thông tin)
3. Tìm kiếm cục bộ

2. Tổng quan về tìm kiếm heuristic

Đối với những bài toán thực tế có không gian trạng thái lớn, tìm kiếm mù sẽ không thực tế do cần phải di chuyển trong không gian tìm kiếm không có định hướng, dẫn đến phải xem xét nhiều trạng thái làm độ phức tạp tính toán và yêu cầu bộ nhớ lớn hơn.

Để giải quyết vấn đề trên, thì "*tìm kiếm heuristic sử dụng thêm thông tin từ bài toán để định hướng tìm kiếm, cụ thể là lựa chọn thứ tự mở rộng nút theo hướng mau dẫn tới đích hơn.*"[1]

Một trong hai chiến lược phổ biến trong tìm kiếm có thông tin là tìm kiếm tham lam (Greedy) và tìm kiếm A*.

Nguyên tắc: sử dụng hàm *f(n)* để đánh giá độ "tốt" tiềm năng của nút n, từ đó chọn nút n có hàm f tốt nhất để mở rộng trước. Thường thì độ tốt được đo bằng giá thành đường đi đến đích, nút có hàm *f(n)* nhỏ hơn sẽ được ưu tiên mở rộng trước. Thông thường, ta chỉ có thể ước lượng hàm *f(n*) dựa vào các thông tin từ bài toán. Hàm *f(n)* thường chứa một thành phần là hàm heuristic *h(n)*, là hàm ước lượng khảng cách từ nút n tới đích.

[1] Từ Minh Phương, Trí Tuệ Nhân Tạo, tr.44

2.1 Thuật toán A*

Nếu như thuật toán tham lam (Greedy) trong tìm kiếm có thông tin không cho lời giải tối ưu (lời giải với đường đi ngắn nhất), do chỉ quan tâm đến khoảng cách ước lượng từ một nút tới đích mà không quan tâm tới quãng đường đã đi từ nút xuất phát tới nút đó, thì thuật toán A* sẽ khắc phục điều này. A* cho lời giải có chi phí nhỏ nhất và cây tìm kiếm có kích thước vừa phải.

Thuật toán A* sẽ sử dụng hàm đánh giá *f(n)* với hai thành phần:

1. Đường đi từ nút đang xét tới nút xuất phát
2. Khoảng cách ước lượng tới đích

Thuật toán A* sử dụng hàm *f(n) = g(n) + h(n):*

1. g(n) = giá thành đường đi từ nút xuất phát đến nút n
2. h(n) = giá thành ước lượng đường đi từ nút n đến nút đích; h(n) là hàm heuristic

Thuật toán A* yêu cầu hàm *h(n)* là hàm chấp nhận được (admissible) theo định nghĩa sau: *Hàm h(n) được gọi là hàm chấp nhận được nếu h(n) không lớn hơn độ dài đường đi thực ngắn nhất từ n tới nút đích.*

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

$$h(n) \leq h^*(n)$$

trong đó h*(n) là giá thành đường đi thực tế từ n đến node đích. Lưu ý rằng hàm h(n)=0 với mọi n, là hàm chấp nhận được.

2.1.1 Mã giả của thuật toán:

A*(Q, S, G, P, c, h)

• Đầu vào: bài toán tìm kiếm, hàm heuristic h

• Đầu ra: đường đi ngắn nhất từ nút xuất phát đến nút đích

• Khởi tạo: tập các nút biên (nút mở) O ← S

While(O không rỗng) do

1. Lấy nút n có f(n) nhỏ nhất ra khỏi O
2. Nếu n thuộc G, return(đường đi tới n)
3. Với mọi m ∈ P(n)
    i. g(m) = g(n) + c(m, n)
    ii. f(m) = g(m) + h(m)
    iii. Thêm m vào O cùng giá trị f(m)

Return: không tìm được đường đi.

2.1.2 Thuật toán A* viết bằng ngôn ngữ Python

Thuật toán A* là một biến thể của giải thuật BFS (Best First Search). Thuật toán BFS sẽ mở rộng cây tìm kiếm theo hướng ưu tiên các nút lá có triển vọng chứa trạng thái đích ((dựa trên hàm đánh giá h).

Giải thuật BFS viết bằng Python:

```
def best_first_graph_search(problem, f, display=False):
    f = memoize(f, 'f')
    node = Node(problem.initial)
    frontier = PriorityQueue('min', f)
    frontier.append(node)
    explored = set()
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            if display:
                print(len(explored), "paths have been expanded and", len(frontier),
"paths remain in the frontier")
            return node
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                frontier.append(child)
            elif child in frontier:
                if f(child) < frontier[child]:
                    del frontier[child]
                    frontier.append(child)
    return None[2]=
```

Thuật toán A* dựa trên BFS:

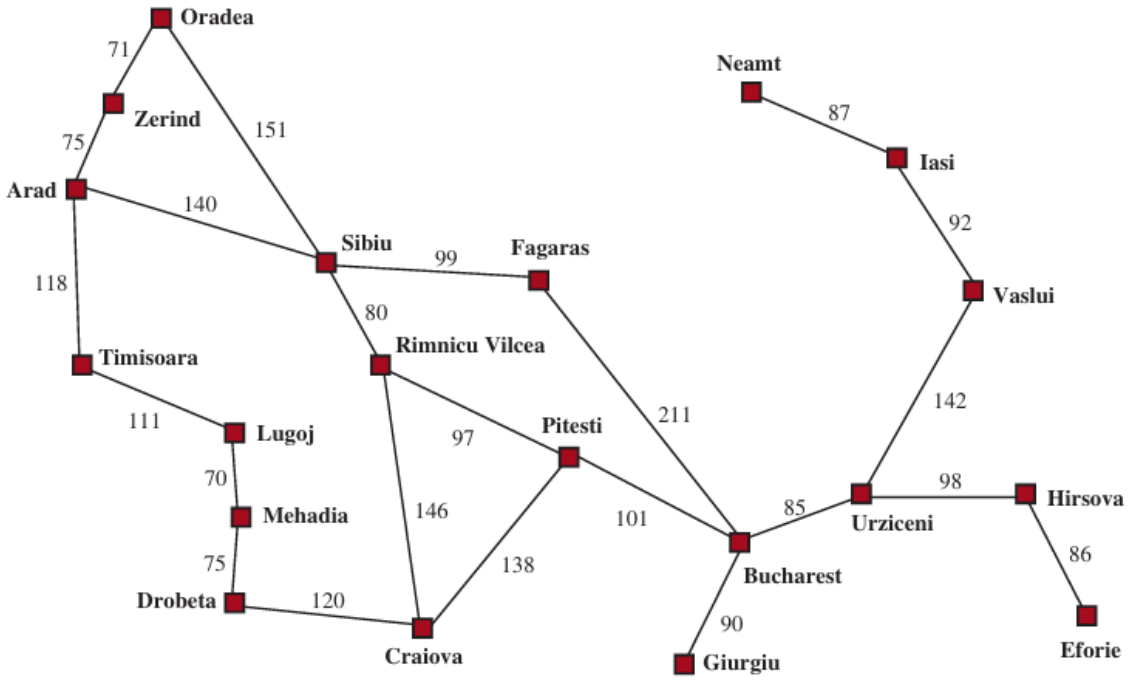
def astar_search(problem, h=None, display=False):
h = memoize(h or problem.h, 'h')
return best_first_graph_search(problem, lambda n: n.path_cost + h(n), display)[3]

Code gốc: https://github.com/aimacode/aima-python/blob/master/search.pyl.

[3] [3] Stuart J. Russell, Peter Norvig, Artificial Intelligence AModern Approach, https://github.com/aimacode/aima-python/blob/master/search.pyl.

CHƯƠNG 2: MỘT SỐ ỨNG DỤNG CỦA THUẬT TOÁN A*

1. Áp dụng thuật toán A* vào bài toán tìm đường đi ngắn nhất:



Hình 1. Một bản đồ đường đơn giản hóa của một phần của Romania, với khoảng cách đường bộ tính bằng dặm.

Cho khoảng cách Ơclit (khoảng cách theo đường chim bay) từ mỗi thành phố đến đích (Bucharest) là:

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Hình 2. Khoảng cách Ơclit (khoảng cách theo đường chim bay) từ mỗi thành phố đến đích (Bucharest)

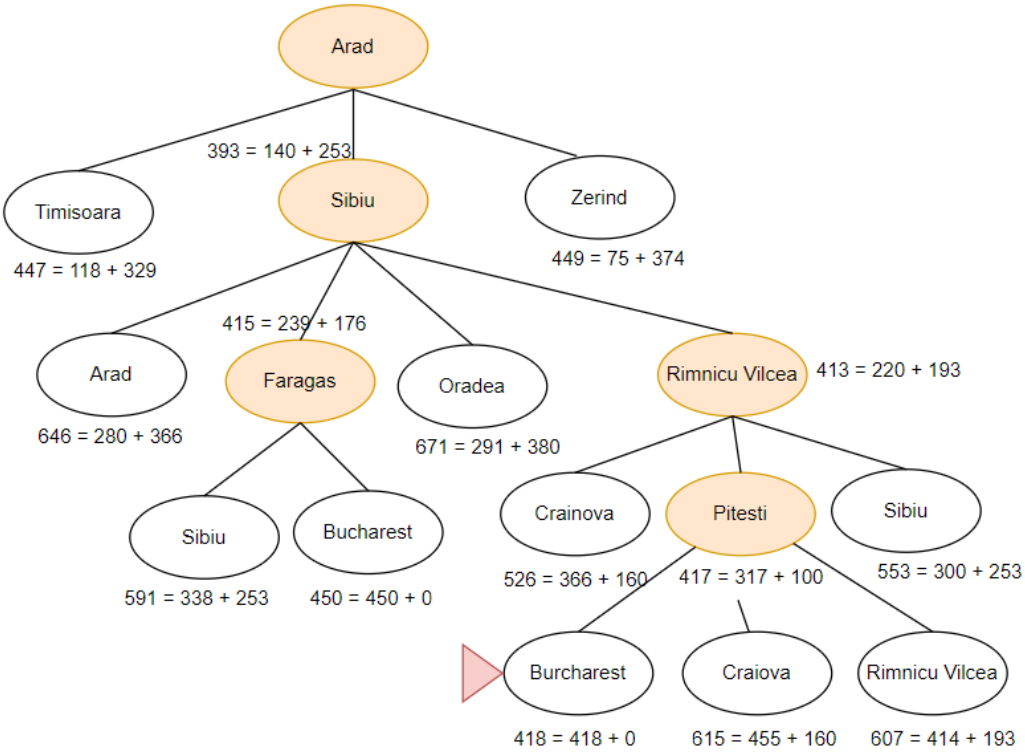
Cần tìm đường đi từ thành phố Arad đến thành phố Bucharest. Lời giải của bài toán là dãy các phép chuyển từ trạng thái đầu đến trạng thái đích, hay là đường đi từ thành phố đầu đến thành phố đích.

Bài toán có thể phát biểu thành 5 thành phần:

- Trạng thái: vị trí của thành phố hiện tại
- Trạng thái đầu: Thành phố Arad
- Trạng thái đích: Thành phố Buchares

- Phép chuyển trạng thái: từ thành phố sang thành phố lân cận
- Chi phí: khoảng cách giữa 2 thành phố trong phép chuyển trạng thái.

Thuật toán A* sẽ tính toán như sau:



Hình 3. Cây tìm kiếm của giải thuật A*

Lời giải bài toán là: Arad – Rimnicu Vilcea – Pitesti - Bucharest.

1.1. Code giao diện GUI:

```
#----
from search import *
import tkinter as tk
import tkinter.ttk as ttk
import time
import numpy as np

romania_map = UndirectedGraph(dict(
   Arad=dict(Zerind=75, Sibiu=140, Timisoara=118),
   Bucharest=dict(Urziceni=85, Pitesti=101, Giurgiu=90, Fagaras=211),
   Craiova=dict(Drobeta=120, Rimnicu=146, Pitesti=138),
   Drobeta=dict(Mehadia=75),
   Eforie=dict(Hirsova=86),
   Fagaras=dict(Sibiu=99),
   Hirsova=dict(Urziceni=98),
   Iasi=dict(Vaslui=92, Neamt=87),
   Lugoj=dict(Timisoara=111, Mehadia=70),
   Oradea=dict(Zerind=71, Sibiu=151),
```

```
    Pitesti=dict(Rimnicu=97),
    Rimnicu=dict(Sibiu=80),
    Urziceni=dict(Vaslui=142)))

romania_map.locations = dict(
    Arad=(91, 492), Bucharest=(400, 327), Craiova=(253, 288),
    Drobeta=(165, 299), Eforie=(562, 293), Fagaras=(305, 449),
    Giurgiu=(375, 270), Hirsova=(534, 350), Iasi=(473, 506),
    Lugoj=(165, 379), Mehadia=(168, 339), Neamt=(406, 537),
    Oradea=(131, 571), Pitesti=(320, 368), Rimnicu=(233, 410),
    Sibiu=(207, 457), Timisoara=(94, 410), Urziceni=(456, 350),
    Vaslui=(509, 444), Zerind=(108, 531))

city_name = dict(
    Arad=(-35, 0), Bucharest=(0, 15), Craiova=(-20, 15),
    Drobeta=(-50, 0), Eforie=(0, 15), Fagaras=(10, 0),
    Giurgiu=(10, 0), Hirsova=(10, 0), Iasi=(10, 0),
    Lugoj=(10, 0), Mehadia=(10, 0), Neamt=(10, -5),
    Oradea=(-50, 0), Pitesti=(-5, 20), Rimnicu=(10, -5),
    Sibiu=(0, -20), Timisoara=(-60, 0), Urziceni=(0, 15),
    Vaslui=(10, 0), Zerind=(-50, 0))

graph_dict = romania_map.graph_dict
romania_locations = romania_map.locations

class App(tk.Tk):
    def  __init__(self):
        super().__init__()

        self.start = 'Arad'
        self.dest = 'Arad'
        self.path_location = None

        self.title('Search')
        self.cvs_map = tk.Canvas(self, width = 640, height = 480,
                                 relief = tk.SUNKEN, border = 1)
        self.ve_ban_do()
        lbl_frm_menu = tk.LabelFrame(self)
        lst_city = []
        for city in city_name:
            lst_city.append(city)

        lbl_start = ttk.Label(lbl_frm_menu, text = 'Start')

        self.cbo_start = ttk.Combobox(lbl_frm_menu, values = lst_city)
        self.cbo_start.set('Arad')
```

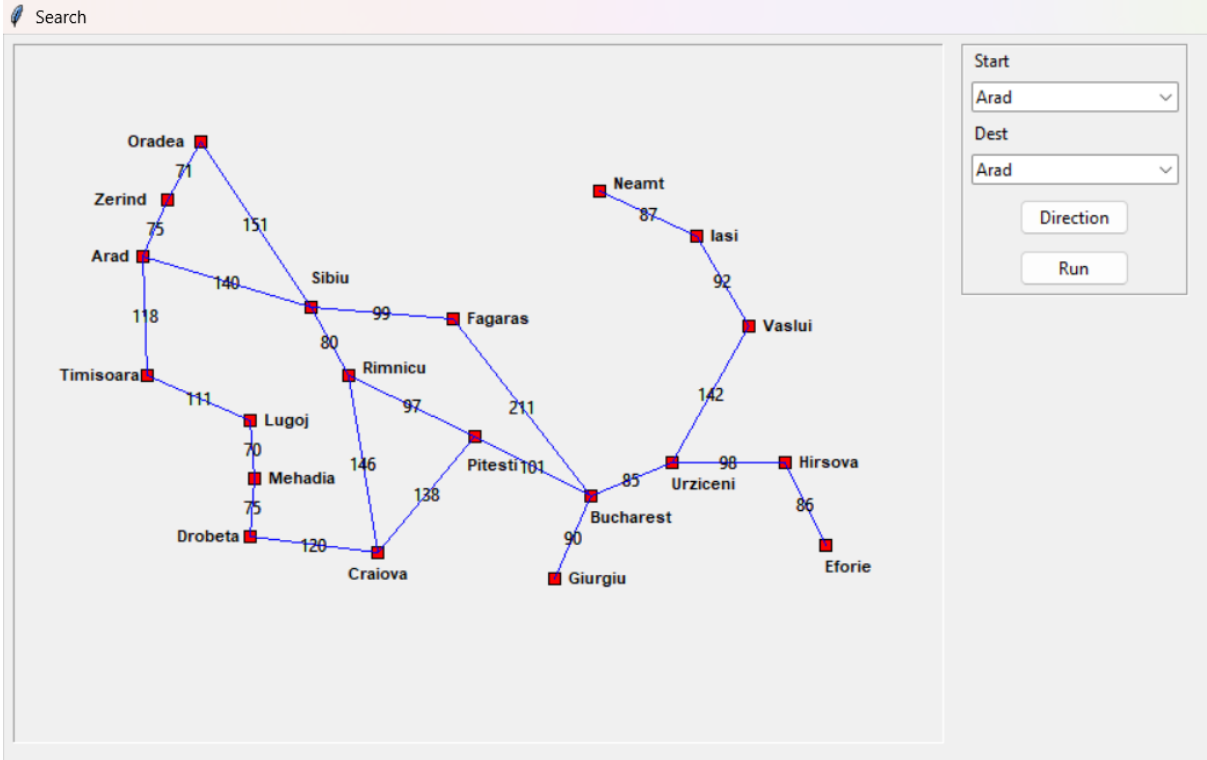
```
        self.cbo_start.bind("<<ComboboxSelected>>", self.cbo_start_click)

        lbl_start.grid(row = 0, column = 0, padx = 5, pady = 0, sticky = tk.W)

        self.cbo_start.grid(row = 1, column = 0, padx = 5, pady = 5)


        lbl_dest = ttk.Label(lbl_frm_menu, text = 'Dest')

        self.cbo_dest = ttk.Combobox(lbl_frm_menu, values = lst_city)
        self.cbo_dest.set('Arad')
        self.cbo_dest.bind("<<ComboboxSelected>>", self.cbo_dest_click)

        btn_direction = ttk.Button(lbl_frm_menu, text = 'Direction',
                           command = self.btn_direction_click)
        btn_run = ttk.Button(lbl_frm_menu, text = 'Run',
                         command = self.btn_run_click)

        lbl_dest.grid(row = 2, column = 0, padx = 5, pady = 0, sticky = tk.W)

        self.cbo_dest.grid(row = 3, column = 0, padx = 5, pady = 5)

        btn_direction.grid(row = 4, column = 0, padx = 5, pady = 5)
        btn_run.grid(row = 5, column = 0, padx = 5, pady = 5)

        self.cvs_map.grid(row = 0, column = 0, padx = 5, pady = 5)
        lbl_frm_menu.grid(row = 0, column = 1, padx = 5, pady = 7, sticky = tk.N)

    def ve_ban_do(self):
        for city in graph_dict:
            x0 = romania_locations[city][0]
            y0 = 640 - romania_locations[city][1]
            self.cvs_map.create_rectangle(x0-4, y0-4, x0+4, y0+4,
                                          fill = 'red', outline = 'black')
            dx = city_name[city][0]
            dy = city_name[city][1]
            self.cvs_map.create_text(x0+dx, y0+dy, text = city, anchor = tk.W, font =
('Time New Romans', 8, 'bold'))

            for neighbor in graph_dict[city]:
                x1 = romania_locations[neighbor][0]
                y1 = 640 - romania_locations[neighbor][1]
                self.cvs_map.create_line(x0, y0, x1, y1,fill = 'blue')
                xm = (x0 + x1)/2
                ym = (y0 + y1)/2
                distance = romania_map.get(city, neighbor)
```

```
            self.cvs_map.create_text(xm, ym, text = str(distance), anchor = tk.CENTER,
fill = 'black')

    def cbo_start_click(self, *args):
        self.start = self.cbo_start.get()

    def cbo_dest_click(self, *args):
        self.dest = self.cbo_dest.get()

    def btn_direction_click(self):

        self.cvs_map.delete(tk.ALL)
        self.ve_ban_do()

        romania_problem = GraphProblem(self.start, self.dest, romania_map)
        c = astar_search(romania_problem)
        lst_path = c.path()
        self.path_location = []
        for data in lst_path:
            city = data.state
            x = romania_map.locations[city][0]
            y = 640 - romania_map.locations[city][1]
            self.path_location.append((x,y))

        self.cvs_map.create_line(self.path_location, fill = 'green')

    def btn_run_click(self):
        bg_color = self.cvs_map['background']
        N = 21
        d = 100
        L = len(self.path_location)
        for i in range(0, L-1):
            x0 = self.path_location[i][0]
            y0 = self.path_location[i][1]
            x1 = self.path_location[i+1][0]
            y1 = self.path_location[i+1][1]
            b = y1-y0
            a = x1-x0
            d1 = np.sqrt((x1-x0)**2 + (y1-y0)**2)
            N1 = int(N*d1/d)
            dt = 1.0/(N1-1)
            for j in range(0,N1):
                t = j*dt
                x = x0 + (x1-x0)*t
                y = y0 + (y1-y0)*t
```

```
            self.cvs_map.delete(tk.ALL)
            self.ve_ban_do()
            self.cvs_map.create_line(self.path_location, fill = 'green')

            self.ve_mui_ten(b,a,x,y,'red')

            time.sleep(0.005)
            self.cvs_map.update()
            self.ve_mui_ten(b,a,x,y,bg_color)
        self.ve_mui_ten(b,a,x,y,'red')


    def ve_mui_ten(self, b, a, tx, ty, color):
        p_mui_ten = [(0,0,1), (-20,10,1), (-15,0,1), (-20,-10,1)]
        p_mui_ten_ma_tran = [np.array([[0],[0],[1]],np.float32),
                             np.array([[-20],[10],[1]],np.float32),
                             np.array([[-15],[0],[1]],np.float32),
                             np.array([[-20],[-10],[1]],np.float32)]

        M1 = np.array([[1, 0, tx],
                       [0, 1, ty],
                       [0, 0, 1]], np.float32)

        theta = np.arctan2(b, a)
        M2 = np.array([[np.cos(theta), -np.sin(theta), 0],
                       [np.sin(theta),  np.cos(theta), 0],
                       [     0,              0,        1]], np.float32)

        M = np.matmul(M1, M2)

        q_mui_ten = []
        for p in p_mui_ten_ma_tran:
            q = np.matmul(M, p)
            q_mui_ten.append((q[0,0], q[1,0]))

        self.cvs_map.create_polygon(q_mui_ten, fill = color, outline = color)

if __name__ == "__main__":
    app =       App()
    app.mainloop()
```

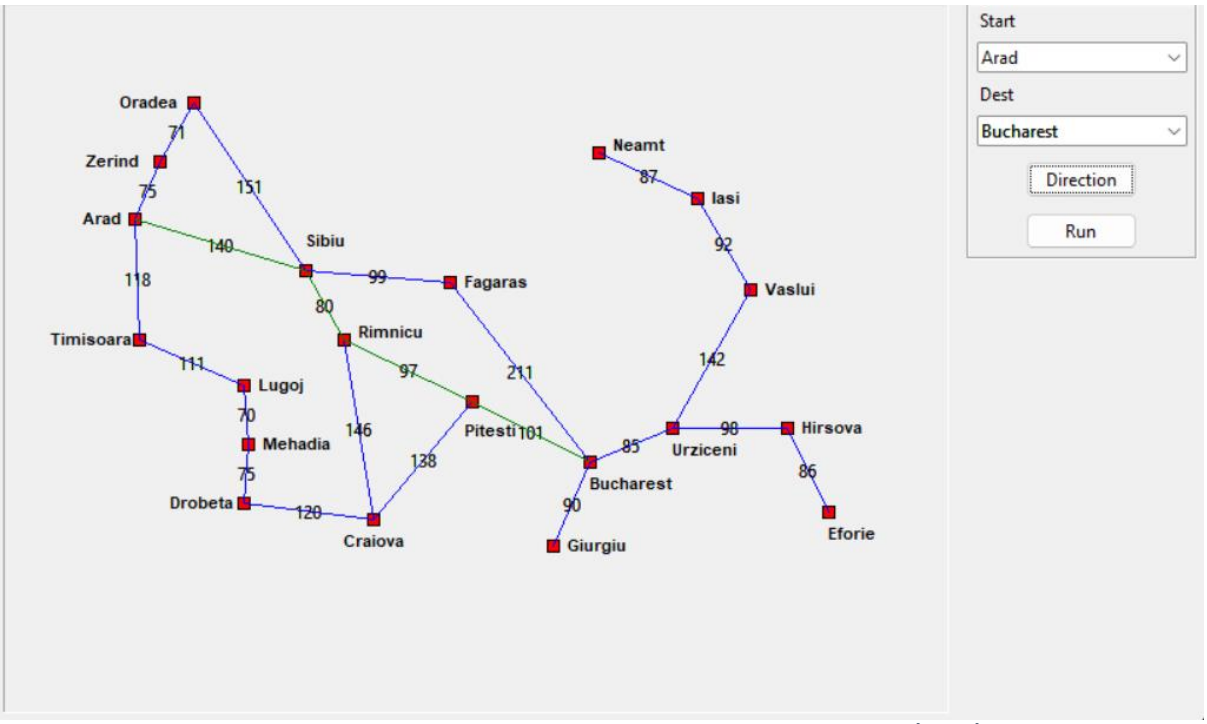
#---------

Kết quả giao diện:



Hình 4. Giao diện GUI của bài toán tìm đường đi ngắn nhất

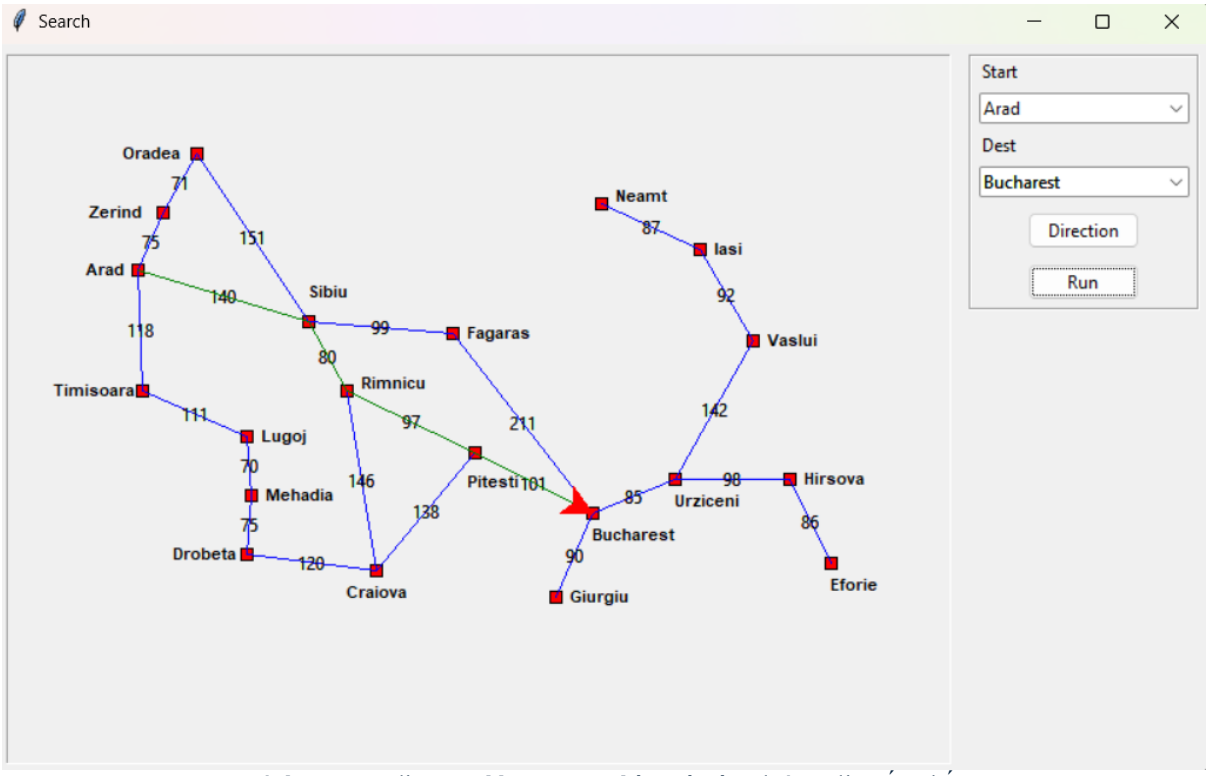
Thông qua lệnh *self.cvs_map.create_line(self.path_location, fill = 'green')* trong hàm *def btn_run_click(self)* thì khi ấn nút "Direction" sẽ vẽ một đường line từ thành phố đầu đến thành phố đích:



Hình 5. Giao diện sau khi Direction của bài toán tìm đường đi ngắn nhất

Thông qua lệnh:
self.ve_mui_ten(b,a,x,y,'red')
time.sleep(0.005)
self.cvs_map.update()
self.ve_mui_ten(b,a,x,y,bg_color)
self.ve_mui_ten(b,a,x,y,'red')
sẽ vẽ nhiều mũi tên màu đỏ chạy từ thành phố đầu đến thành phố đích, sau đó cập nhật lại màu sắc của mũi tên bằng màu 'bg_color' để xóa các mũi tên đã vẽ.

Khi ấn nút "Run", sẽ có mũi tên động chạy trên đường line từ thành phố đầu đến thành phố đích:



Hình 6. Giao diện sau khi Run của bài toán tìm đường đi ngắn nhất

1.2. Code giao diện web

#---------

```
import streamlit as st
import streamlit.components.v1 as components
import time
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np
from search import *

romania_map = UndirectedGraph(dict(
   Arad=dict(Zerind=75, Sibiu=140, Timisoara=118),
```

```
    Bucharest=dict(Urziceni=85, Pitesti=101, Giurgiu=90, Fagaras=211),
    Craiova=dict(Drobeta=120, Rimnicu=146, Pitesti=138),
    Drobeta=dict(Mehadia=75),
    Eforie=dict(Hirsova=86),
    Fagaras=dict(Sibiu=99),
    Hirsova=dict(Urziceni=98),
    Iasi=dict(Vaslui=92, Neamt=87),
    Lugoj=dict(Timisoara=111, Mehadia=70),
    Oradea=dict(Zerind=71, Sibiu=151),
    Pitesti=dict(Rimnicu=97),
    Rimnicu=dict(Sibiu=80),
    Urziceni=dict(Vaslui=142)))

romania_map.locations = dict(
    Arad=(91, 492), Bucharest=(400, 327), Craiova=(253, 288),
    Drobeta=(165, 299), Eforie=(562, 293), Fagaras=(305, 449),
    Giurgiu=(375, 270), Hirsova=(534, 350), Iasi=(473, 506),
    Lugoj=(165, 379), Mehadia=(168, 339), Neamt=(406, 537),
    Oradea=(131, 571), Pitesti=(320, 368), Rimnicu=(233, 410),
    Sibiu=(207, 457), Timisoara=(94, 410), Urziceni=(456, 350),
    Vaslui=(509, 444), Zerind=(108, 531))
city_name = dict(
    Arad=(-35, 0), Bucharest=(0, 15), Craiova=(-20, 15),
    Drobeta=(-50, 0), Eforie=(0, 15), Fagaras=(10, 0),
    Giurgiu=(10, 0), Hirsova=(10, 0), Iasi=(10, 0),
    Lugoj=(10, 0), Mehadia=(10, 0), Neamt=(10, -5),
    Oradea=(-50, 0), Pitesti=(-5, 20), Rimnicu=(10, -5),
    Sibiu=(0, -20), Timisoara=(-60, 0), Urziceni=(0, 15),
    Vaslui=(10, 0), Zerind=(-50, 0))
map_locations = romania_map.locations
graph_dict = romania_map.graph_dict
lst_city = []
for city in city_name:
    lst_city.append(city)
xmin = 91
xmax = 562
ymin = 270
ymax = 570
def ve_ban_do():
    fig, ax = plt.subplots()
    ax.axis([xmin-70, xmax+70, ymin-70, ymax+70])
    for key in graph_dict:
        city = graph_dict[key]
        x0 = map_locations[key][0]
        y0 = map_locations[key][1]
```

```
        rectangle = plt.Rectangle((x0-4, y0-4), 8, 8, edgecolor='black',color = 'r' ,
linewidth=1.0)
        ax.add_patch(rectangle)
        dx = city_name[key][0]
        dy = city_name[key][1]
        ten = ax.text(x0+dx,y0-dy,key,fontsize = 6)
        for neighbor in city:
            x1 = map_locations[neighbor][0]
            y1 = map_locations[neighbor][1]
            xm = (x0 + x1)/2 + 5
            ym = (y0 + y1)/2 + 5
            distance = romania_map.get(key, neighbor)
            if (distance == 71 or distance == 75):
                ax.text(xm-3, ym-10, str(distance), fontsize = 6)
            elif(distance == 101 or distance == 85):
                ax.text(xm-10, ym, str(distance), fontsize = 6)
            else:
                ax.text(xm, ym, str(distance), fontsize = 6)
            doan_thang, = ax.plot([x0, x1], [y0, y1], 'blue', linewidth = 1)
    return fig

def ve_mui_ten(b, a, tx, ty):
    p_mui_ten = [(0,0,1), (-20,10,1), (-15,0,1), (-20,-10,1)]
    p_mui_ten_ma_tran = [np.array([[0],[0],[1]],np.float32),
                         np.array([[-20],[10],[1]],np.float32),
                         np.array([[-15],[0],[1]],np.float32),
                         np.array([[-20],[-10],[1]],np.float32)]

    # Tạo ma trận dời (tịnh tiến) - translate
    M1 = np.array([[1, 0, tx],
                   [0, 1, ty],
                   [0, 0, 1]], np.float32)

    # Tạo ma trận quay - rotation
    theta = np.arctan2(b, a)
    M2 = np.array([[np.cos(theta), -np.sin(theta), 0],
                   [np.sin(theta),  np.cos(theta), 0],
                   [     0,              0,       1]], np.float32)

    M = np.matmul(M1, M2)

    q_mui_ten = []

    for p in p_mui_ten_ma_tran:
        q = np.matmul(M, p)
        q_mui_ten.append([q[0,0], q[1,0]])
```

```
    return q_mui_ten

if "flag_anim" not in st.session_state:
    st.session_state["flag_anim"] = False
if st.session_state["flag_anim"] == False:
    if "flag_ve_ban_do" not in st.session_state:
        st.session_state["flag_ve_ban_do"] = True
        fig = ve_ban_do()
        st.session_state['fig'] = fig
        st.pyplot(fig)
        print(st.session_state["flag_ve_ban_do"])
        print('Vẽ bản đồ lần đầu')
    else:
        if st.session_state["flag_ve_ban_do"] == False:
            st.session_state["flag_ve_ban_do"] = True
            fig = ve_ban_do()
            st.session_state['fig'] = fig
            st.pyplot(fig)
        else:
            print('Đã vẽ bản đồ')
            st.pyplot(st.session_state['fig'])
    lst_city = []
    for city in city_name:
        lst_city.append(city)

    start_city = st.selectbox('Bạn chọn thành phố bắt đầu:', lst_city)
    dest_city = st.selectbox('Bạn chọn thành phố đích:', lst_city)

    st.session_state['start_city'] = start_city
    st.session_state['dest_city']  = dest_city

    if st.button('Direction'):
        romania_problem = GraphProblem(start_city, dest_city, romania_map)
        c = astar_search(romania_problem)
        lst_path = c.path()
        print('Con duong tim thay: ')

        for data in lst_path:
            city = data.state
            print(city, end = ' ')
        print()
        path_locations = {}
        for data in lst_path:
            city = data.state
            path_locations[city] = map_locations[city]
        print(path_locations)
```

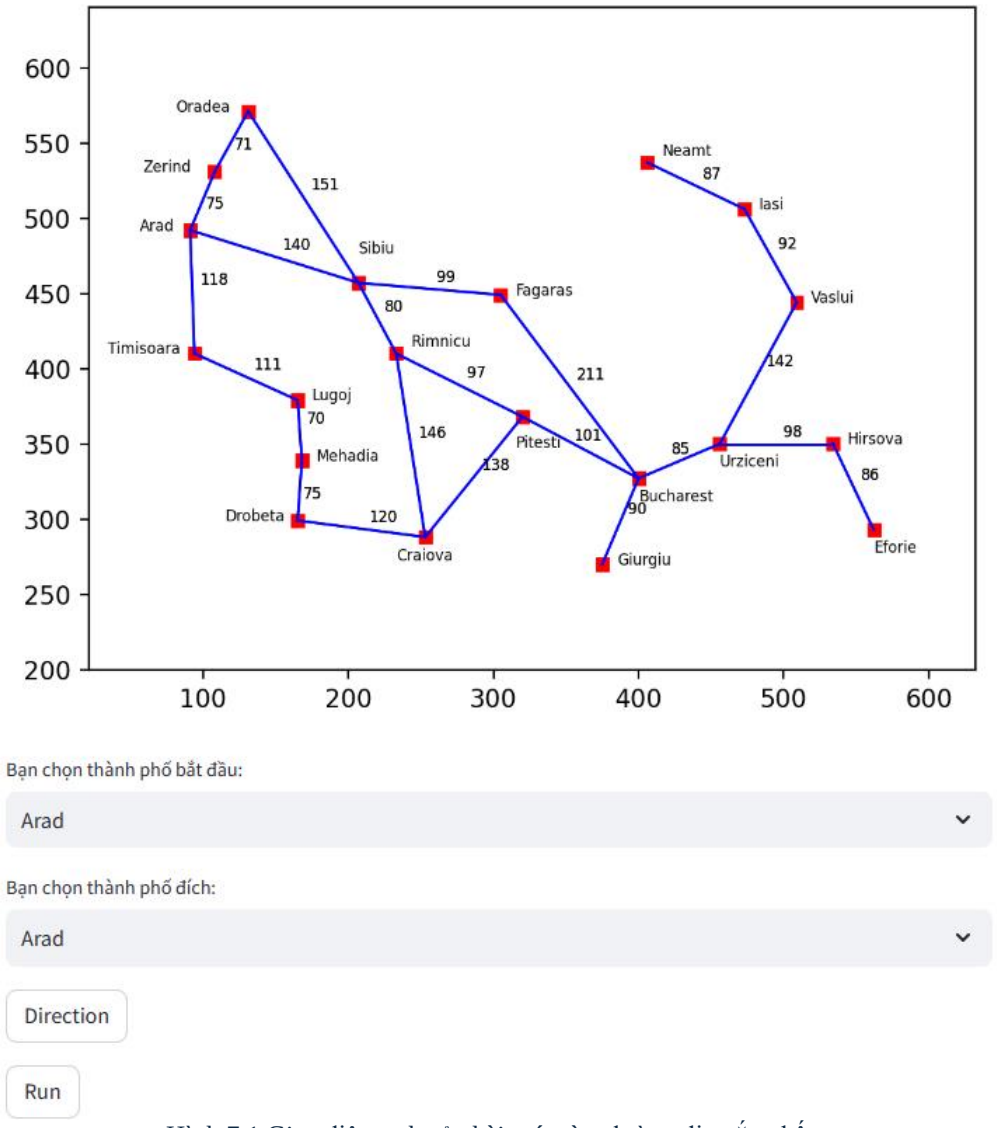
```
        lst_path_location_x = []
        lst_path_location_y = []

        for city in path_locations:
            lst_path_location_x.append(path_locations[city][0])
            lst_path_location_y.append(path_locations[city][1])

        print(lst_path_location_x)
        print(lst_path_location_y)


        fig, ax = plt.subplots()
        ax.axis([xmin-70, xmax+70, ymin-70, ymax+70])

        for key in graph_dict:
            city = graph_dict[key]
            x0 = map_locations[key][0]
            y0 = map_locations[key][1]

            rectangle = plt.Rectangle((x0-4, y0-4), 8, 8, edgecolor='black',color = 'r' ,
linewidth=1.0)
            ax.add_patch(rectangle)

            dx = city_name[key][0]
            dy = city_name[key][1]
            ten = ax.text(x0+dx,y0-dy,key, fontsize = 6)

            for neighbor in city:
                x1 = map_locations[neighbor][0]
                y1 = map_locations[neighbor][1]
                xm = (x0 + x1)/2 + 5
                ym = (y0 + y1)/2 + 5
                distance = romania_map.get(key, neighbor)
                if (distance == 71 or distance == 75):
                    ax.text(xm-3, ym-10, str(distance), fontsize = 6)
                elif(distance == 101 or distance == 85):
                    ax.text(xm-10, ym, str(distance), fontsize = 6)
                else:
                    ax.text(xm, ym, str(distance), fontsize = 6)
                doan_thang, = ax.plot([x0, x1], [y0, y1], 'b')
            path_tim_thay, = ax.plot(lst_path_location_x, lst_path_location_y, 'green')
        print('Đã gán fig có hướng dẫn')
        st.session_state['fig'] = fig
        st.rerun()
    if st.button('Run'):
```

```
    start_city = st.session_state['start_city']
    dest_city = st.session_state['dest_city']

    romania_problem = GraphProblem(start_city, dest_city, romania_map)
    c = astar_search(romania_problem)
    lst_path = c.path()
    print('Con duong tim thay: ')

    for data in lst_path:
        city = data.state
        print(city, end = ' ')
    print()
    path_locations = {}
    for data in lst_path:
        city = data.state
        path_locations[city] = map_locations[city]
    print(path_locations)

    lst_path_location_x = []
    lst_path_location_y = []

    for city in path_locations:
        lst_path_location_x.append(path_locations[city][0])
        lst_path_location_y.append(path_locations[city][1])

    print(lst_path_location_x)
    print(lst_path_location_y)

    fig, ax = plt.subplots()

    dem = 0

    lst_doan_thang = []
    for key in graph_dict:
        city = graph_dict[key]
        x0 = map_locations[key][0]
        y0 = map_locations[key][1]

        rectangle = plt.Rectangle((x0-4, y0-4), 8, 8, edgecolor='black',color = 'r' ,
linewidth=1.0)
        ax.add_patch(rectangle)
        lst_doan_thang.append(rectangle)

        dx = city_name[key][0]
        dy = city_name[key][1]
        ten = ax.text(x0+dx,y0-dy,key)
```

```
    lst_doan_thang.append(ten)

    for neighbor in city:
        x1 = map_locations[neighbor][0]
        y1 = map_locations[neighbor][1]
        xm = (x0 + x1)/2 + 5
        ym = (y0 + y1)/2 + 5
        distance = romania_map.get(key, neighbor)

        if (distance == 71 or distance == 75):
            ax.text(xm-3, ym-10, str(distance), fontsize = 6)
        elif(distance == 101 or distance == 85):
            ax.text(xm-10, ym, str(distance), fontsize = 6)
        else:
            ax.text(xm, ym, str(distance), fontsize = 6)
        doan_thang, = ax.plot([x0, x1], [y0, y1], 'b')
        lst_doan_thang.append(doan_thang)
        dem = dem + 1

    path_tim_thay, = ax.plot(lst_path_location_x, lst_path_location_y, 'green')
    lst_doan_thang.append(path_tim_thay)

print('Dem: ', dem)
lst_vi_tri = []
N = 21
d = 100
L = len(lst_path_location_x)
for i in range(0,L-1):
    x1 = lst_path_location_x[i]
    y1 = lst_path_location_y[i]
    x2 = lst_path_location_x[i+1]
    y2 = lst_path_location_y[i+1]

    d0 = np.sqrt((x2-x1)**2 + (y2-y1)**2)
    N0 = int(N*d0/d)
    dt = 1/(N0-1)
    for j in range(0, N0):
        t = j*dt
        x = x1 + (x2-x1)*t
        y = y1 + (y2-y1)*t
        q = ve_mui_ten(y2-y1, x2-x1, x, y)
        lst_vi_tri.append(q)

red_polygon, = ax.fill([],[], color = 'red')

FRAME = len(lst_vi_tri)
```

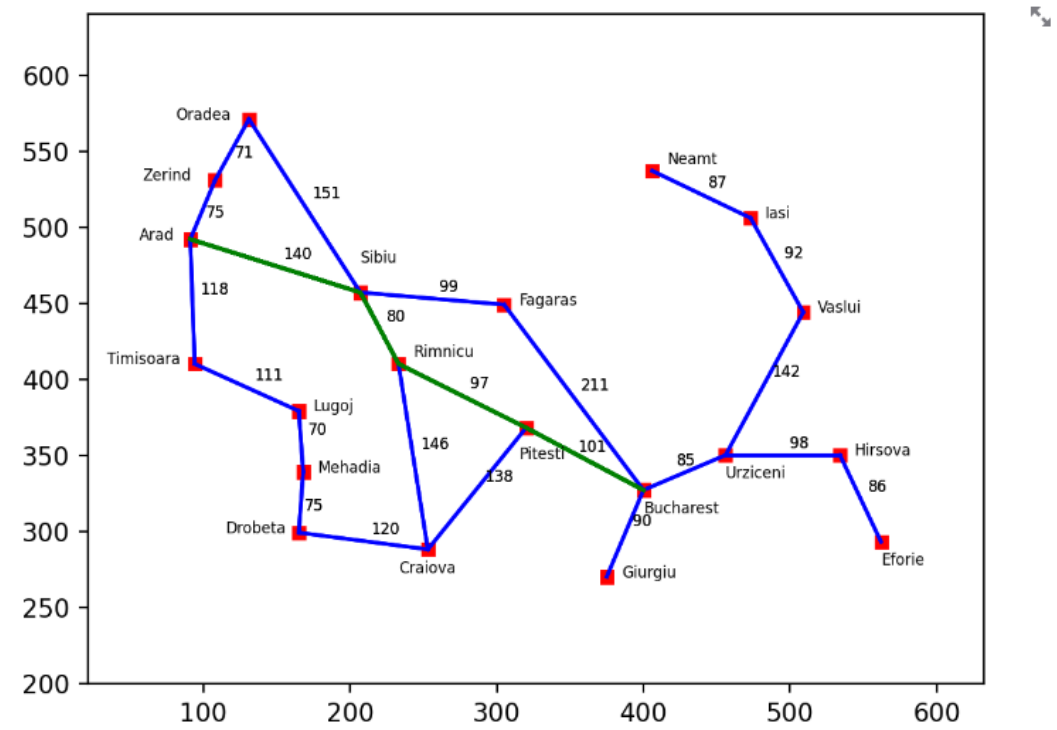
```
        def init():
            ax.axis([xmin-70, xmax+70, ymin-70, ymax+70])
            return lst_doan_thang, red_polygon

        def animate(i):
            red_polygon.set_xy(lst_vi_tri[i])
            return lst_doan_thang, red_polygon

        anim = FuncAnimation(fig, animate, frames= FRAME, interval=50,
init_func=init, repeat=False)

        st.session_state["flag_anim"] = True
        st.session_state['anim'] = anim
        st.rerun()

else:
    if st.session_state["flag_anim"] == True:
        components.html(st.session_state["anim"].to_jshtml(), height = 550)
        if st.button('Reset'):
            st.session_state["flag_anim"] = False
            st.session_state["flag_ve_ban_do"] = False
            st.rerun()
                                                #----
```

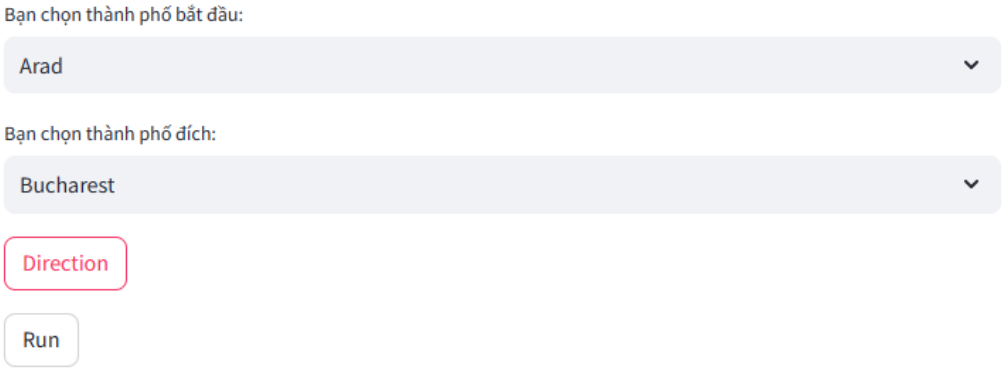
Kết quả giao diện:



Hình 7.1 Giao diện web của bài toán tìm đường đi ngắn nhất

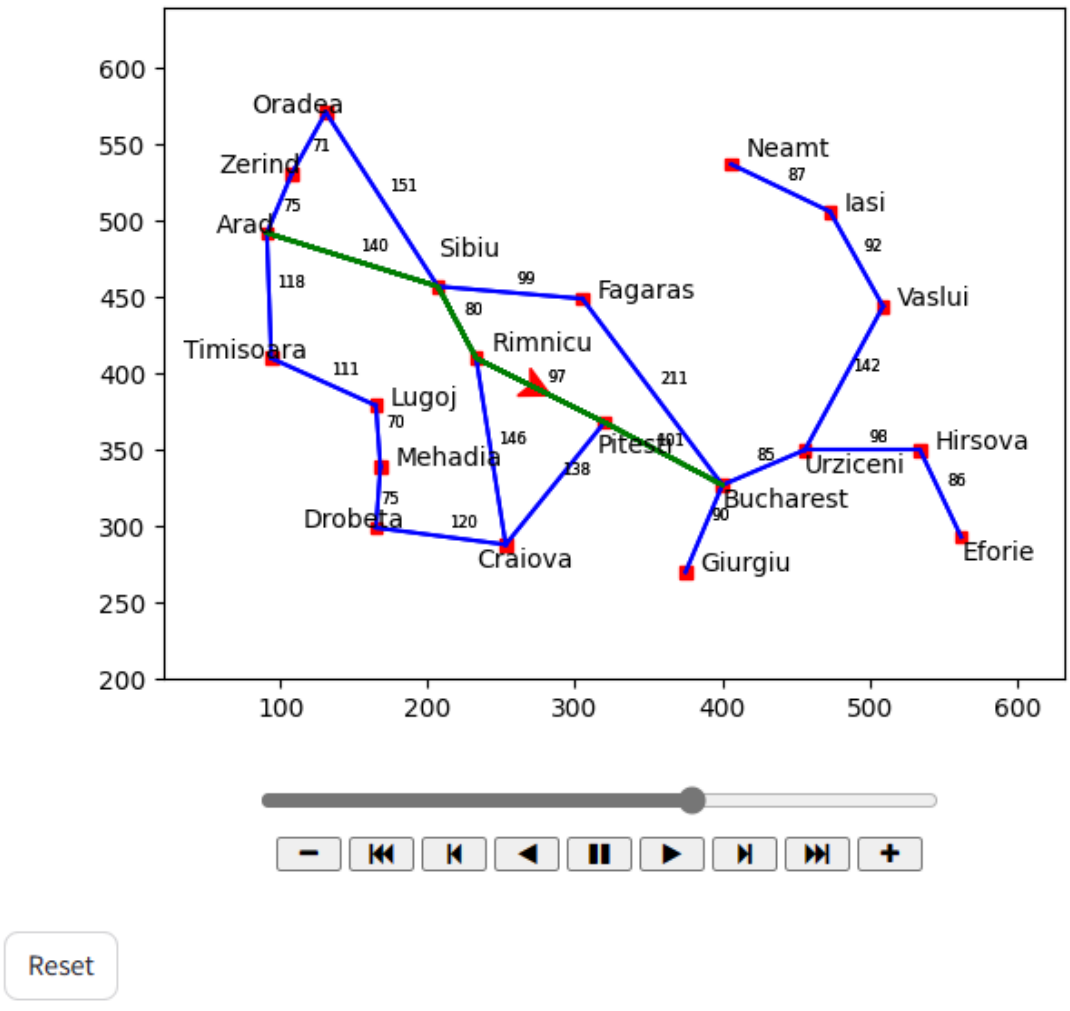
Sau khi bấm nút “Direction”:





Hình 7.2 Giao diện web của bài toán tìm đường đi ngắn nhất

Sau khi bấm nút “Run”:



Hình 7.3 Giao diện web của bài toán tìm đường đi ngắn nhất

2. Ứng dụng thuật toán A* vào bài toán tìm đường trong mê cung

Cho 1 mê cung, mỗi vị trí biểu diễn bởi ‘#’ là tường của mê cung, không thể đi vào, vị trí biểu diễn là đường đi, có thể đi được, ‘o’ là điểm khởi đầu , ‘x’ là điểm cuối. Tìm đường đi ngắn nhất từ vị trí bắt đầu đến vị trí kết thúc.

Xác định chi phí di chuyển xung quanh mê cung. Di chuyển chéo đắt hơn di chuyển ngang hoặc dọc:

Di chuyển chéo = 1.7

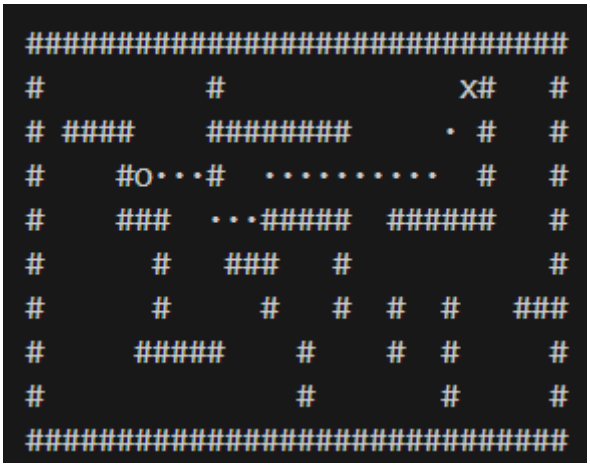
Di chuyển dọc = 1.0

Bài toán được mô tả trong không gian trạng thái với các đặc điểm sau:

- Trạng thái ban đầu : các ô mang giá trị ‘#’ biểu diễn cho bức tường trong mê cung.

- Các thao tác/ hành động để tạo ra các trạng thái mới, di chuyển hướng đi lên trên, xuống dưới, qua trái, qua phải và đánh dấu là đoạn đường đi.
- Trạng thái bắt đầu: Khởi tạo ma trận mê cung, có ô bắt đầu và ô đích đến. ‘o’ là điểm khởi đầu , ‘x’ là điểm cuối.
- Trạng thái đích: Đường đi từ ô bắt đầu đến ô đích.Việc tìm giải pháp cho bài toán là tìm đường đi từ ô bắt đầu đến ô đích.

Ví dụ một lời giải của bài toán:



Hình 8. Ví dụ lời giải mê cung in ra màn hình

Chúng ta sử dụng khoảng cách Euclidean cho việc tính toán hàm heuristic trong trường hợp này:

$$h(n) = \sqrt{(x - gx)^2 + (y - gy)^2}$$

Trong đó: (x,y) tương ứng với tọa độ ô hiện tại trong mê cung còn (gx, gy) là tọa độ ô đích đến. Sử dụng thuật toán A* sẽ tính khoảng cách ngắn nhất di chuyển từ ‘o’ đến ‘x’ dựa trên hàm heuristic. Từ đó, đưa ra những di chuyển phù hợp cho đường đi đến đích.

2.1 Code giao diện GUI

```
#-----
import math
from simpleai.search import SearchProblem, astar
import numpy as np
import cv2
import tkinter as tk
import tkinter.ttk as ttk
from PIL import Image, ImageTk
import time
from tkinter import messagebox
```

```
from matplotlib.animation import FuncAnimation
# Define cost of moving around the map
cost_regular = 1.0
cost_diagonal = 1.7

# Create the cost dictionary
COSTS = {
    "up": cost_regular,
    "down": cost_regular,
    "left": cost_regular,
    "right": cost_regular,
    "up left": cost_diagonal,
    "up right": cost_diagonal,
    "down left": cost_diagonal,
    "down right": cost_diagonal,
}
# Define the map
MAP =  """
##################################
#       #          #  #
# ####   ########     #  #
#   #   #          #  #
#   ###    ##### ######  #
#     #  ###  #        #
#     #    #  # # #  ###
#    #####   #   # #    #
#          #     #    #
##################################

"""

MAP = [list(x) for x in MAP.split("\n") if x]
M = 10
N = 30
W = 21
mau_xanh = np.zeros((W, W, 3), np.uint8) + (np.uint8(255), np.uint8(0), np.uint8(0))
mau_trang = np.zeros((W, W, 3), np.uint8) + (np.uint8(255), np.uint8(255),
np.uint8(255))
image = np.ones((M*W, N*W, 3), np.uint8)*255

for x in range (0, M):
    for y in range (0,N):
        if MAP[x][y] == "#":
            image[x*W:(x+1)*W, y*W:(y+1)*W] = mau_xanh
        elif MAP[x][y] == ' ':
            image[x*W:(x+1)*W, y*W:(y+1)*W] = mau_trang
```

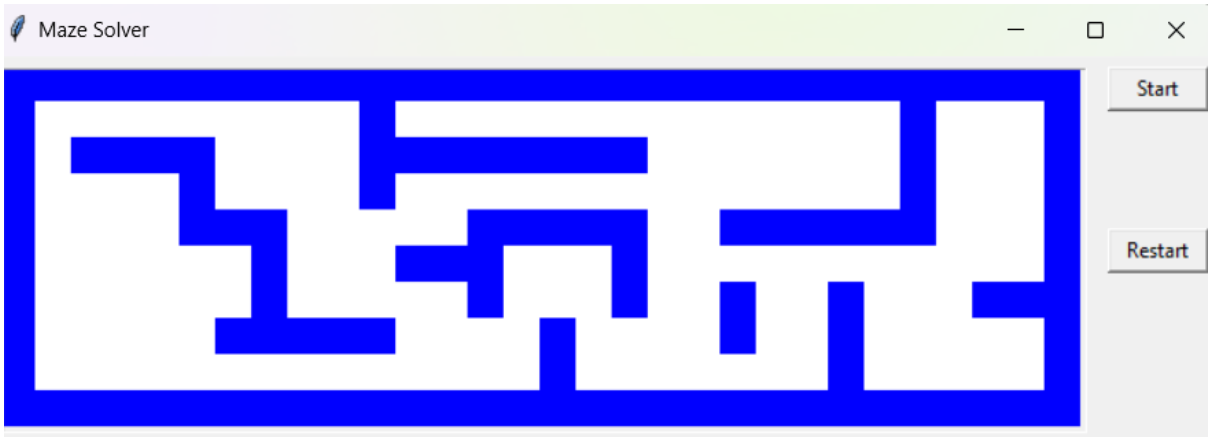
```
color_coverted = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
pil_image = Image.fromarray(color_coverted)

class MazeSolver(SearchProblem):
    # Initialize the class
    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)

        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)
        super(MazeSolver, self).__init__(initial_state=self.initial)
    def actions(self, state):
        actions = []
        for action in COSTS.keys():
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
                actions.append(action)
        return actions

    # Update the state based on the action
    def result(self, state, action):
        x, y = state

        if action.count("up"):
            y -= 1
        if action.count("down"):
            y += 1
        if action.count("left"):
            x -= 1
        if action.count("right"):
            x += 1

        new_state = (x, y)

        return new_state

    # Check if we have reached the goal
    def is_goal(self, state):
        return state == self.goal

    # Compute the cost of taking an action
```

```
    def cost(self, state, action, state2):
        return COSTS[action]

    # Heuristic that we use to arrive at the solution
    def heuristic(self, state):
        x, y = state
        gx, gy = self.goal
        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.dem = 0
        self.title("Maze Solver")
        self.cvs_me_cung = tk.Canvas(self, width=N*W, height=M*W, relief =
tk.SUNKEN, border = 1)
        self.image_tk = ImageTk.PhotoImage(pil_image)
        self.cvs_me_cung.create_image(0, 0, anchor = tk.NW, image = self.image_tk)
        self.cvs_me_cung.bind("<Button-1>", self.xu_ly_mouse)
        btn_start  = tk.Button(self, text = "Start", width = 7, command =
self.btn_start_click)
        btn_restart = tk.Button(self, text = "Restart", width = 7, command =
self.btn_restart_click)

        self.cvs_me_cung.grid(row = 0, column = 0, padx = 5, pady = 5)
        btn_start.grid(row = 0, column = 1, padx = 5, pady = 6, sticky = tk.N) #pady la
khoang cach giua 2 button
        btn_restart.grid(row = 0, column = 1, padx = 5, pady = 6)

    def btn_start_click(self):
        # Create a map
        problem = MazeSolver(MAP)
        # Run the search
        result = astar(problem, graph_search=True)
        # Extract the path
        if result is not None:
            path = [x[1] for x in result.path()]
        else:
            print("No path found.")
            messagebox.showinfo("Thông báo", "Không tìm thấy đường đi")
            path = []
        # Print the result
        print()
        for y in range(len(MAP)):
            for x in range(len(MAP[y])):
                if (x, y) == problem.initial:
```

```
                print('o', end='')
            elif (x, y) == problem.goal:
                print('x', end='')
            elif (x, y) in path:
                print('·', end='')
            else:
                print(MAP[y][x], end='')

        print()
    print(path)
    L = len(path)
    bg_color = self.cvs_me_cung['background']  # Define the variable "bg_color"
    for i in range(1, L):
        x1, y1 = path[i-1]
        x2, y2 = path[i]
        line = self.cvs_me_cung.create_line(x1*W+W//2, y1*W+W//2, x2*W+W//2,
y2*W+W//2, fill='red', width=2)
        arrow = self.ve_mui_ten(y2-y1, x2-x1, x2*W+W//2, y2*W+W//2, 'red')
        self.cvs_me_cung.update()
        time.sleep(0.1)
        self.cvs_me_cung.delete(arrow)
        self.cvs_me_cung.update()

    self.ve_mui_ten(y2-y1, x2-x1, x2*W+W//2, y2*W+W//2, 'red')

def ve_mui_ten(self, b, a, tx, ty, color):
    p_mui_ten = [(0,0,1), (-20,10,1), (-15,0,1), (-20,-10,1)]
    p_mui_ten_ma_tran = [np.array([[0],[0],[1]],np.float32),
                         np.array([[-20],[10],[1]],np.float32),
                         np.array([[-15],[0],[1]],np.float32),
                         np.array([[-20],[-10],[1]],np.float32)]

    M1 = np.array([[1, 0, tx],
                   [0, 1, ty],
                   [0, 0, 1]], np.float32)

    theta = np.arctan2(b, a)
    M2 = np.array([[np.cos(theta), -np.sin(theta), 0],
                   [np.sin(theta),  np.cos(theta), 0],
                   [     0,              0,        1]], np.float32)

    M = np.matmul(M1, M2)

    q_mui_ten = []
    for p in p_mui_ten_ma_tran:
        q = np.matmul(M, p)
```

```
            q_mui_ten.append((q[0,0], q[1,0]))
        arrow = self.cvs_me_cung.create_polygon(q_mui_ten, fill = color, outline =
color)
        return arrow
    def btn_restart_click(self):

        # Delete all items from the canvas
        self.cvs_me_cung.delete('all')
        # Redraw the initial image
        self.cvs_me_cung.create_image(0, 0, anchor = tk.NW, image = self.image_tk)

        self.dem = 0
        self.cvs_me_cung.update()
        for y in range(len(MAP)):
            for x in range(len(MAP[y])):
                if MAP[y][x] == 'o':
                    MAP[y][x] = ' '
                elif MAP[y][x] == 'x':
                    MAP[y][x] = ' '
        self.cvs_me_cung.bind("<Button-1>", self.xu_ly_mouse)
    def xu_ly_mouse(self, event):
        if self.dem == 0:
            px = event.x
            py = event.y
            x = px//W
            y = py//W
            MAP[y][x] = 'o'
            self.cvs_me_cung.create_oval(x*W+2, y*W+2, (x+1)*W-2,(y+1)*W-2, outline
= '#FF0000', fill = '#FF0000', width = 2)
            self.dem += 1
        elif self.dem == 1:
            px = event.x
            py = event.y
            x = px//W
            y = py//W
            MAP[y][x] = 'x'
            self.cvs_me_cung.create_rectangle(x*W+2, y*W+2, (x+1)*W-2,(y+1)*W-2,
outline = '#FF0000', fill = '#FF0000', width = 2)
            self.dem += 1

if __name__ == "__main__":
    app = App()
    app.mainloop()
                                                #-------
```

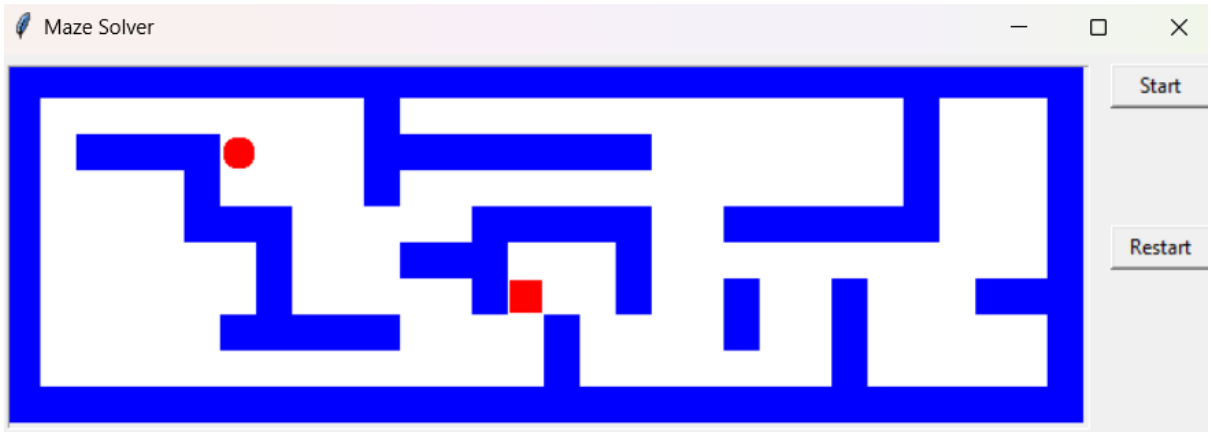
Kết quả giao diện: Giao diện hiện lên gồm có 2 phần chính: Bên phải là phần Option, bên trái là mê cung được xây dựng sẵn, nơi thực hiện visualize thuật toán.



Hình 9.1 Giao diện bài toán tìm đường trong mê cung

Lựa chọn các options: Người dùng bắt đầu chọn node source và destination source trực tiếp trên mê cung và bắt đầu thuật toán. Node source là hình tròn được tô màu đỏ, node destination là hình vuông được tô màu đỏ.

Chọn vị trí bắt đầu và vị trí kết thúc trực tiếp trên mê cung qua hàm *def xu_ly_mouse(self, event):*



Hình 9.2 Giao diện bài toán tìm đường trong mê cung

Sau khi source được chọn thì lần click tiếp theo được hiểu là chọn destination, sau khi cả source và destination được chọn thì bấm "Start" thuật toán sẽ tự động tiến hành.

Khi ấn nút "Start", thuật toán sẽ vẽ đường line và mũi tên theo đường đi "path" tìm được từ thuật toán A* thông qua lệnh:

```
for i in range(1, L):
        x1, y1 = path[i-1]
        x2, y2 = path[i]
```

*line = self.cvs_me_cung.create_line(x1*W+W//2, y1*W+W//2, x2*W+W//2, y2*W+W//2, fill='red', width=2)*

*arrow = self.ve_mui_ten(y2-y1, x2-x1, x2*W+W//2, y2*W+W//2, 'red')*

Xóa các mũi tên đã vẽ:

self.cvs_me_cung.update()

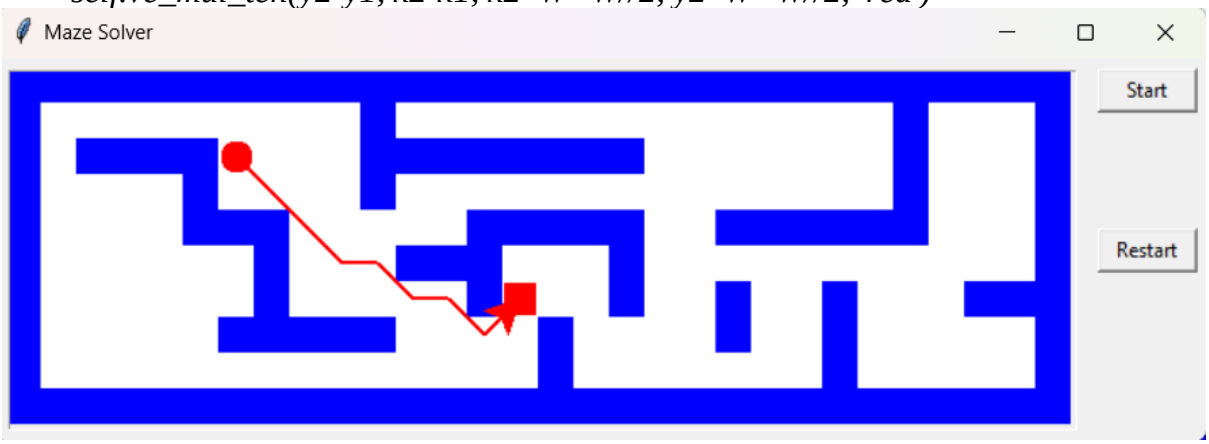
time.sleep(0.1)

self.cvs_me_cung.delete(arrow)

self.cvs_me_cung.update()

Vẽ mũi tên ở đích đến:

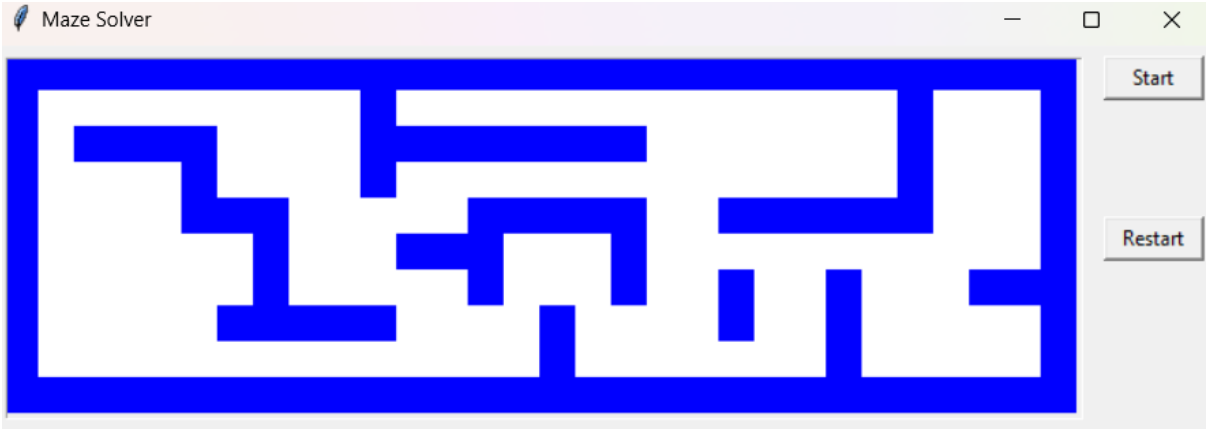
*self.ve_mui_ten(y2-y1, x2-x1, x2*W+W//2, y2*W+W//2, 'red')*



Hình 9.3 Giao diện bài toán tìm đường trong mê cung

Khi ấn nút "Restart", sẽ khởi tạo lại mê cung ban đầu, xóa vị trí của 'x' và 'o' trong mê cung:

```
self.cvs_me_cung.delete('all')
    # Redraw the initial image
    self.cvs_me_cung.create_image(0, 0, anchor = tk.NW, image = self.image_tk)

    self.dem = 0
    self.cvs_me_cung.update()
    for y in range(len(MAP)):
        for x in range(len(MAP[y])):
            if MAP[y][x] == 'o':
                MAP[y][x] = ' '
            elif MAP[y][x] == 'x':
                MAP[y][x] = ' '
    self.cvs_me_cung.bind("<Button-1>", self.xu_ly_mouse)
```



Hình 9.4 Giao diện bài toán tìm đường trong mê cung

2.2. Code giao diện web

```
                                                    #-----
import streamlit as st
import math
from search import *
from PIL import Image, ImageTk
import cv2
from matplotlib.animation import FuncAnimation
import numpy as np
import matplotlib.pyplot as plt
# Define cost of moving around the map
cost_regular = 1.0
cost_diagonal = 1.7

# Create the cost dictionary
COSTS = {
    "up": cost_regular,
    "down": cost_regular,
    "left": cost_regular,
    "right": cost_regular,
    "up left": cost_diagonal,
    "up right": cost_diagonal,
    "down left": cost_diagonal,
    "down right": cost_diagonal,
}

# Define the map
MAP = """
##################################
#       #          #  #
# ####   ########     #  #
#   #   #          #  #
#   ###    ##### ######  #
#     #  ###  #        #
```

```
#    #   #  # # #  ###
#   #####   #   # #   #
#         #     #   #
################################
"""
MAP = MAP.split("\n")

x_start = st.slider("x_start", 1, 28, 2)
y_start = st.slider("y_start", 2, 9, 2)
x_end = st.slider("x_end", 1, 28, 24)
y_end = st.slider("y_end", 2, 9, 7)

MAP[y_start] = MAP[y_start][:x_start] + "o" + MAP[y_start][x_start+1:]
MAP[y_end] = MAP[y_end][:x_end] + "x" + MAP[y_end][x_end+1:]
MAP = "\n".join(MAP)

MAP = [list(x) for x in MAP.split("\n") if x]
M = 10
N = 30
W = 21
mau_xanh = np.zeros((W, W, 3), np.uint8) + (np.uint8(0), np.uint8(0), np.uint8(255))
mau_trang = np.zeros((W, W, 3), np.uint8) + (np.uint8(255), np.uint8(255),
np.uint8(255))
image = np.ones((M*W, N*W, 3), np.uint8)*255
st.title("Maze Solver")
for x in range (0, M):
   for y in range (0,N):
      if MAP[x][y] == "#":
         image[x*W:(x+1)*W, y*W:(y+1)*W] = mau_xanh
      elif MAP[x][y] == ' ':
         image[x*W:(x+1)*W, y*W:(y+1)*W] = mau_trang
      elif MAP[x][y].lower() == 'o':
         center_coordinates = (y*W + W//2, x*W + W//2)
         radius = W//2
         color = (255, 0, 0)
         thickness = -1
         image = cv2.circle(image, center_coordinates, radius, color, thickness)
      elif MAP[x][y].lower() == 'x':
         center_coordinates = (y*W + W//2, x*W + W//2)
         radius = W//2
         color = (0, 0, 255)
         thickness = -1
         image = cv2.rectangle(image, (y*W, x*W), (y*W + W, x*W + W), (255,0,0), -
1)
pil_image = Image.fromarray(image)
st.image(pil_image)
```

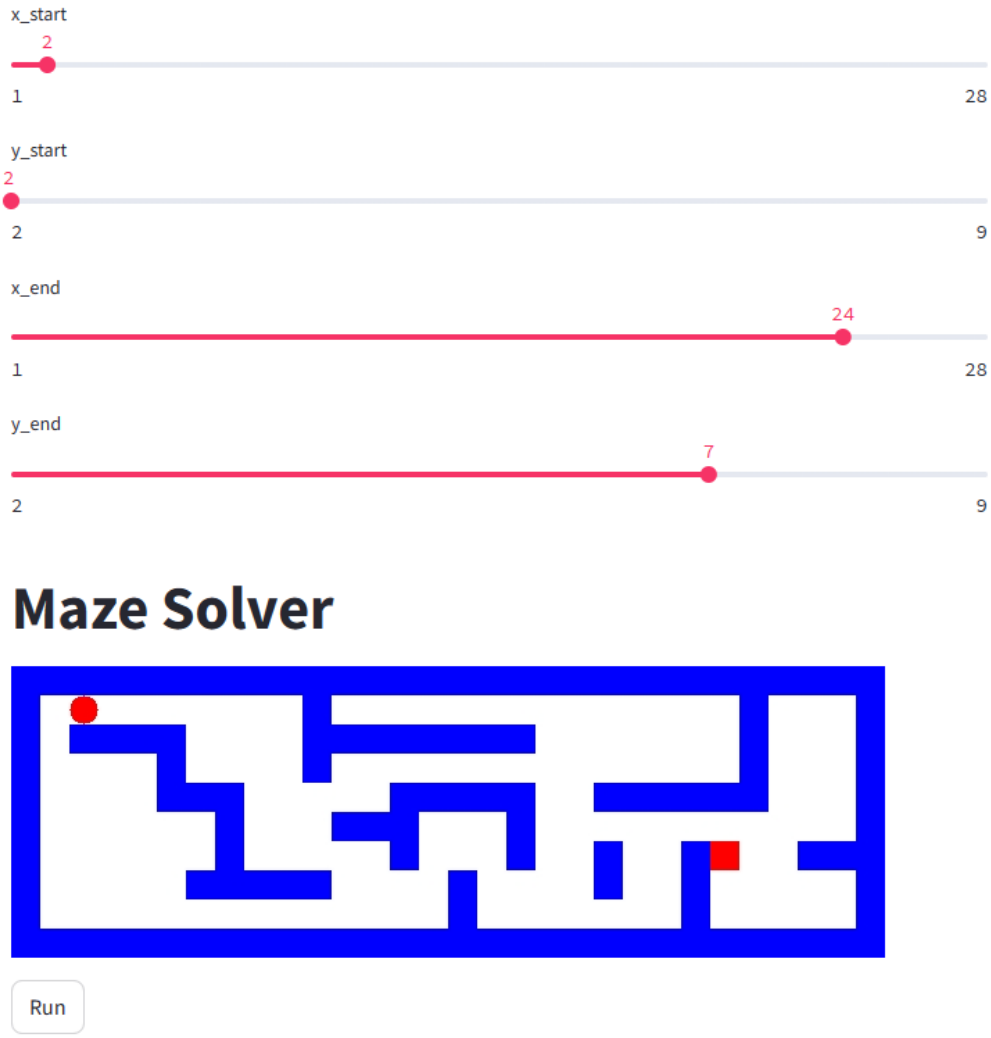
```
def ve_mui_ten(b, a, tx, ty):
    p_mui_ten = [(0,0,1), (-20,10,1), (-15,0,1), (-20,-10,1)]
    p_mui_ten_ma_tran = [np.array([[0],[0],[1]],np.float32),
                         np.array([[-20],[10],[1]],np.float32),
                         np.array([[-15],[0],[1]],np.float32),
                         np.array([[-20],[-10],[1]],np.float32)]

    # Tạo ma trận dời (tịnh tiến) - translate
    M1 = np.array([[1, 0, tx],
                   [0, 1, ty],
                   [0, 0, 1]], np.float32)

    # Tạo ma trận quay - rotation
    theta = np.arctan2(b, a)
    M2 = np.array([[np.cos(theta), -np.sin(theta), 0],
                   [np.sin(theta),  np.cos(theta), 0],
                   [     0,              0,        1]], np.float32)

    M = np.matmul(M1, M2)

    q_mui_ten = []

    for p in p_mui_ten_ma_tran:
        q = np.matmul(M, p)
        q_mui_ten.append([q[0,0], q[1,0]])
    return q_mui_ten




class MazeSolver(Problem):
    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)
        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)
    def actions(self, state):
        actions = []
        for action in COSTS.keys():
            newx, newy = self.result(state, action)
            if self.board[newy][newx] != "#":
```

```
            actions.append(action)
        return actions
    def result(self, state, action):
        x, y = state
        if action.count("up"):
            y -= 1
        if action.count("down"):
            y += 1
        if action.count("left"):
            x -= 1
        if action.count("right"):
            x += 1
        new_state = (x, y)
        return new_state
    def is_goal(self, state):
        return state == self.goal
    def path_cost(self, c, state1, action, state2):
        return c + COSTS[action]
    def h(self, node):
        (x, y) = node.state
        (gx, gy) = self.goal
        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)

if st.button("Run"):
    problem = MazeSolver(MAP)
    result = astar_search(problem)
    path = [x.state for x in result.path()]
    for i in range(0, len(path) - 1):
        pt1 = (path[i][0]*W + W//2, path[i][1]*W + W//2)
        pt2 = (path[i+1][0]*W + W//2, path[i+1][1]*W + W//2)
        image = cv2.line(image, pt1, pt2, (255, 0, 0), 2)
    pil_image = Image.fromarray(image)
    for y in range(len(MAP)):
        for x in range(len(MAP[y])):
            if (x, y) == problem.initial:
                print('o', end='')
            elif (x, y) == problem.goal:
                print('x', end='')
            elif (x, y) in path:
                print('.', end='')
            else:
                print(MAP[y][x], end='')
        print()

    fig, ax = plt.subplots()
    ax.imshow(image)
```

```
    L = len(path)
    lst_vi_tri = []

    for i in range(0, L - 1):
        x1, y1 = path[i]
        x2, y2 = path[i + 1]
        q = ve_mui_ten(y2-y1, x2-x1, x2*W+W//2, y2*W+W//2)
        lst_vi_tri.append(q)

    red_polygon, = ax.fill([],[], 'r')

    FRAME = len(lst_vi_tri)
    def init():
        ax.axis([0, N*W, M*W, 0])
        return red_polygon,
    def animate(i):
        red_polygon.set_xy(lst_vi_tri[i])
        return red_polygon,
    anim = FuncAnimation(fig, animate, frames= FRAME, interval=50, init_func=init,
repeat=False)
    anim.save('animation.gif', writer='imagemagick')
    st.image('animation.gif')
```

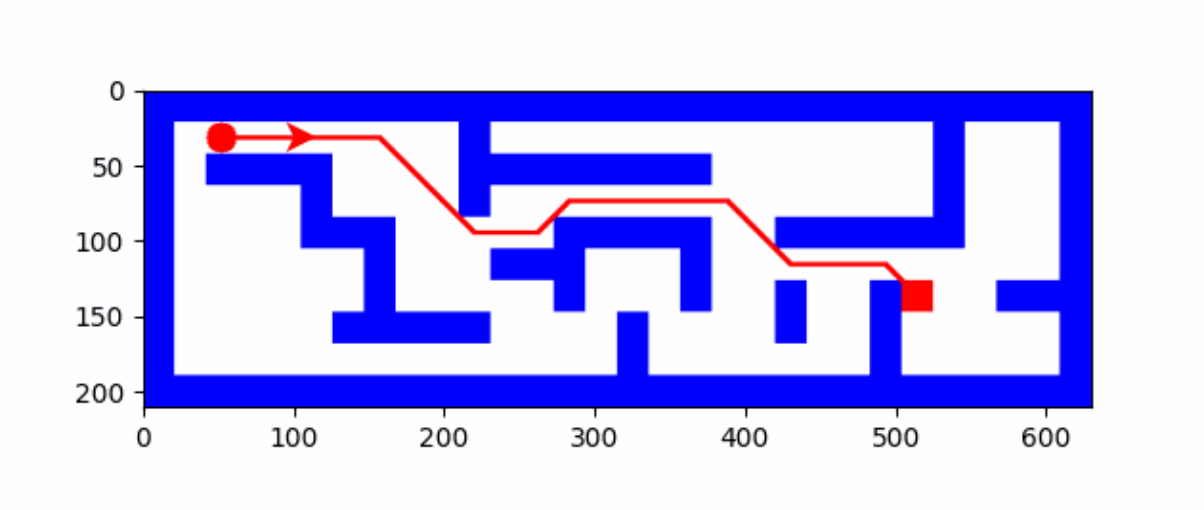
#----------

Kết quả giao diện: Thanh slider dùng để điều chỉnh vị trí tọa độ của ‘o’ (x_start, y_start) và ‘x’ (x_end, y_end) trên mê cung:



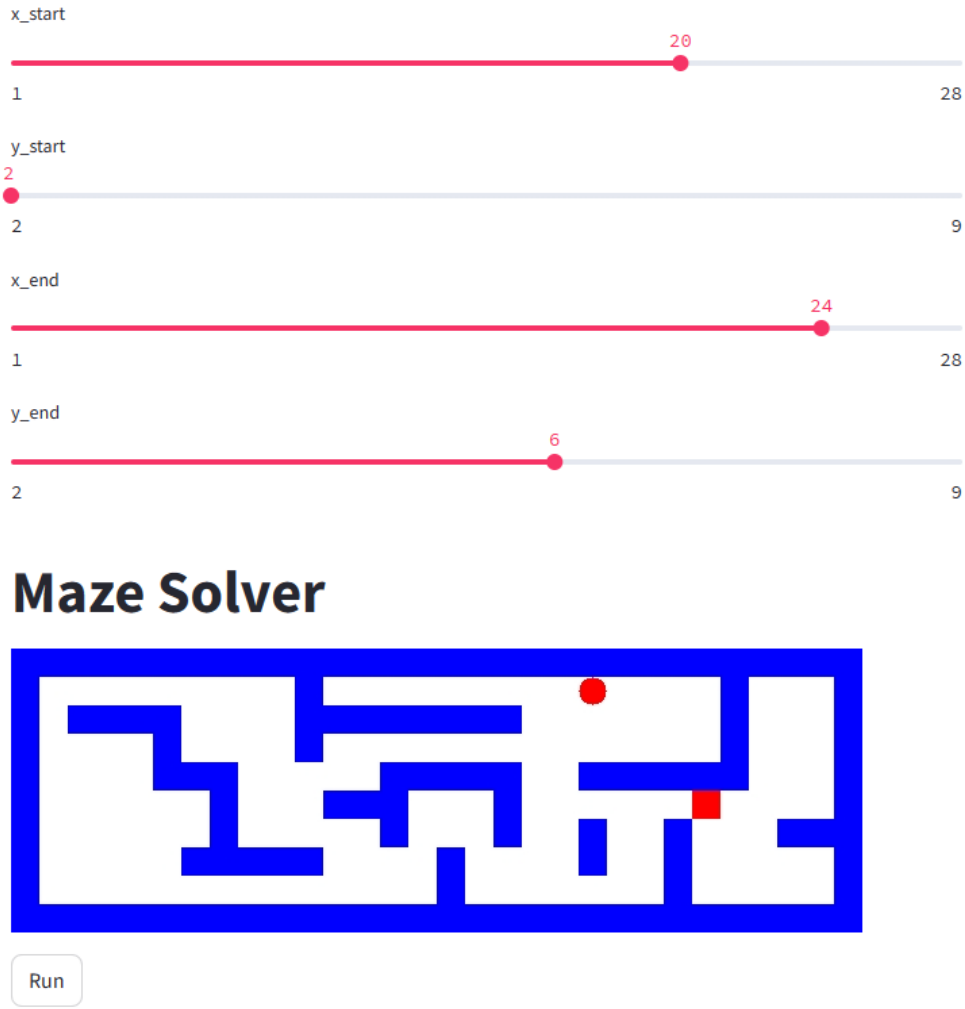
Hình 10. Giao diện web của bài toán tìm đường trong mê cung

Khi bấm nút “Run”, sẽ có đường line nối từ vị trí ‘o’ đến vị trí ‘x’, tức đường đi của bài toán, và mũi tên động chạy theo đường line này:



Hình 11. Giao diện web của bài toán tìm đường trong mê cung

Khi điều chỉnh thanh slider thì giao diện reset lại như trước. Không cần dùng đến button “Reset” để điều chỉnh lại trạng thái ban đầu của mê cung:



Hình 12. Giao diện web của bài toán tìm đường trong mê cung

3. Ứng dụng thuật toán A* vào bài toán 8-Puzzle và bài toán xếp ảnh:

Bài toán 8 ô:

Trạng thái hiện thời u:

3	1	6
5		4
2	7	8

Trạng thái đích:

1	2	3
4	5	6
7	8	9

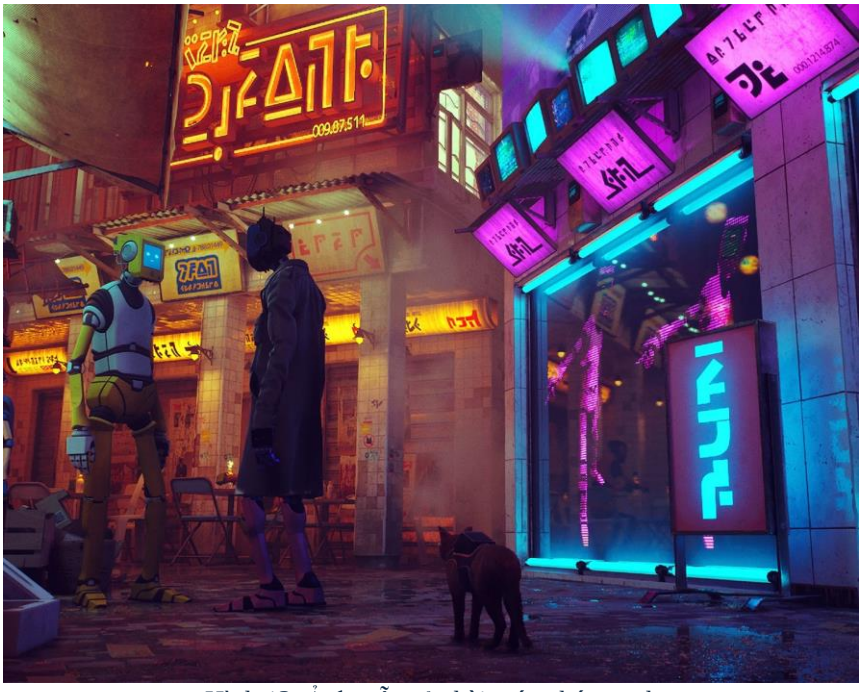
Trong bài toán 8 ô, có thể xây dựng một số hàm heuristic:

h(n) tính bằng tổng khoảng cách Manhattan giữa vị trí hiện thời của mỗi ô tới vị trí đúng của ô đó. Khoảng cách Manhattan được tính bằng số ít nhất các dịch chuyển theo hàng hoặc cột để đưa một quân tới vị trí của nó trong trạng thái đích. Ví dụ, để đưa quân 2 tới vị trí đích ta cần 4 dịch chuyển và do vậy khoảng cách Manhattan của 2 tới đích là 4. Giá trị h của trạng thái trên hình đầu tiên sẽ bằng h(u) = 1 + 4 + 1 + 2 + 1. Hàm h cũng là hàm chấp nhận được.

Ý tưởng bài toán xếp tranh: Ứng dụng cách giải của 8-Puzzle, cắt hình ảnh thành 8 mảnh ghép, gán mỗi mảnh ghép ứng với 1 state = [1, 2, 3, 4 ,5 , 6 ,7, 0] trong thuật toán (dùng dictionary). Hình ảnh thứ 9 sẽ là một ảnh màu trắng. Sau khi "Run" sẽ tạo được ảnh hoàn chỉnh.

Bài toán sử dụng hình ảnh mẫu tên "stray-2.jpg":

image = cv2.imread('stray-2.jpg')



Hình 13. Ảnh mẫu của bài toán ghép tranh

3.1 Code giao diện GUI 8-Puzzle:

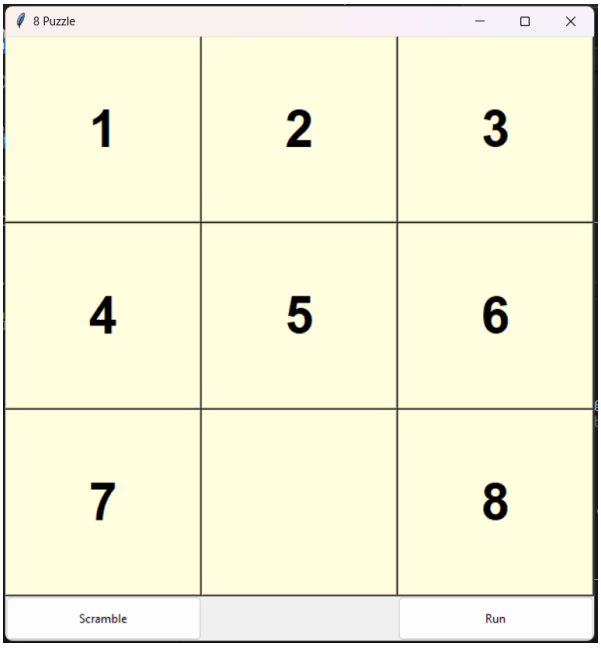
```
import os.path
import random
import time
from functools import partial
from tkinter import *
from tkinter import ttk
import tkinter as tk

from search import astar_search, EightPuzzle
import sys

sys.path.append(os.path.join(os.path.dirname(__file__), '..'))

root = Tk()

state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
puzzle = EightPuzzle(tuple(state))
solution = None

b = [None] * 9
root.title('8 Puzzle')
puzzle_gui = tk.Canvas (root, width = 640, height = 480, relief = tk.SUNKEN, border = 1)
# TODO: refactor into OOP, remove global variables

def scramble():
    global state
    global puzzle
    possible_actions = ['UP', 'DOWN', 'LEFT', 'RIGHT']
    scramble = []
```

```
    for _ in range(60):
        scramble.append(random.choice(possible_actions))
    for move in scramble:
        if move in puzzle.actions(state):
            state = list(puzzle.result(state, move))
            puzzle = EightPuzzle(tuple(state))
            create_buttons()
def solve():
    return astar_search(puzzle).solution()
def solve_steps():

    global puzzle
    global solution
    global state
    solution = solve()
    print(solution)
    for move in solution:
        state = puzzle.result(state, move)
        create_buttons()
        root.update()
        root.after(1, time.sleep(0.75))

def exchange(index):
    """Interchanges the position of the selected tile with the zero tile under certain
conditions"""
    global state
    global solution
    global puzzle
    zero_ix = list(state).index(0)
    actions = puzzle.actions(state)
    current_action = ''
    i_diff = index // 3 - zero_ix // 3
    j_diff = index % 3 - zero_ix % 3
    if i_diff == 1:
        current_action += 'DOWN'
    elif i_diff == -1:
        current_action += 'UP'
    if j_diff == 1:
        current_action += 'RIGHT'
    elif j_diff == -1:
        current_action += 'LEFT'

    if abs(i_diff) + abs(j_diff) != 1:
        current_action = ''

    if current_action in actions:
        b[zero_ix].grid_forget()
        b[zero_ix] = Button(root, text=f'{state[index]}', width=6, font=('Helvetica', 40, 'bold'),
bg='lightyellow',
                            command=partial(exchange, zero_ix))
```

```
        b[zero_ix].grid(row=zero_ix // 3, column=zero_ix % 3, ipady=40)
        b[index].grid_forget()
        b[index] = Button(root, text=None, width=6, font=('Helvetica', 40, 'bold'),
command=partial(exchange, index))
        b[index].grid(row=index // 3, column=index % 3, ipady=40)
        state[zero_ix], state[index] = state[index], state[zero_ix]
        puzzle = EightPuzzle(tuple(state))

def create_buttons():
    # TODO: Find a way to use grid_forget() with a for loop for initialization
    b[0] = Button(root, text=f'{state[0]}' if state[0] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 0))
    b[0].grid(row=0, column=0, ipady=40)
    b[1] = Button(root, text=f'{state[1]}' if state[1] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg= 'lightyellow',
                  command=partial(exchange, 1))
    b[1].grid(row=0, column=1, ipady=40)
    b[2] = Button(root, text=f'{state[2]}' if state[2] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 2))
    b[2].grid(row=0, column=2, ipady=40)
    b[3] = Button(root, text=f'{state[3]}' if state[3] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'),bg='lightyellow',
                  command=partial(exchange, 3))
    b[3].grid(row=1, column=0, ipady=40)
    b[4] = Button(root, text=f'{state[4]}' if state[4] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 4))
    b[4].grid(row=1, column=1, ipady=40)
    b[5] = Button(root, text=f'{state[5]}' if state[5] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 5))
    b[5].grid(row=1, column=2, ipady=40)
    b[6] = Button(root, text=f'{state[6]}' if state[6] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 6))
    b[6].grid(row=2, column=0, ipady=40)
    b[7] = Button(root, text=f'{state[7]}' if state[7] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 7))
    b[7].grid(row=2, column=1, ipady=40)
    b[8] = Button(root, text=f'{state[8]}' if state[8] != 0 else None, width=6,
font=('Helvetica', 40, 'bold'), bg='lightyellow',
                  command=partial(exchange, 8))
    b[8].grid(row=2, column=2, ipady=40)

def create_static_buttons():
    scramble_btn = ttk.Button(root, text='Scramble', width=8, command=partial(scramble))
    scramble_btn.grid(row=3, column=0, ipady=10, sticky=tk.EW)
```

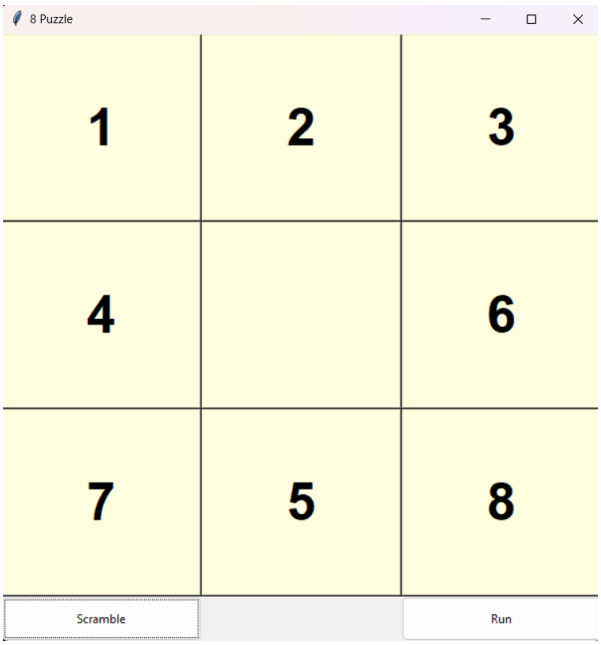
```
    run_btn = ttk.Button(root, text='Run', width=8, command=partial(solve_steps))
    run_btn.grid(row=3, column=2, ipady=10, sticky=tk.EW)

def init():
    global state
    global solution
    state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    scramble()
    create_buttons()
    create_static_buttons()
init()
root.mainloop()
```

Kết quả giao diện:



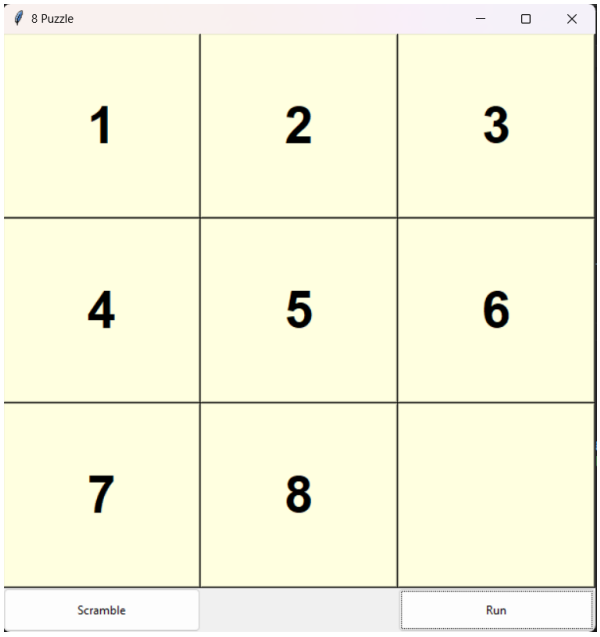
Hình 14. Giao diện GUI của bài toán 8-Puzzle

Sau khi "Scamble":



Hình 15. Giao diện GUI của bài toán 8-Puzzle

Sau khi “Run”:



Hình 16. Giao diện GUI của bài toán 8-Puzzle

Kết quả in ra màn hình:
['DOWN', 'RIGHT']

3.2 Code giao diện GUI bài toán xếp ảnh:

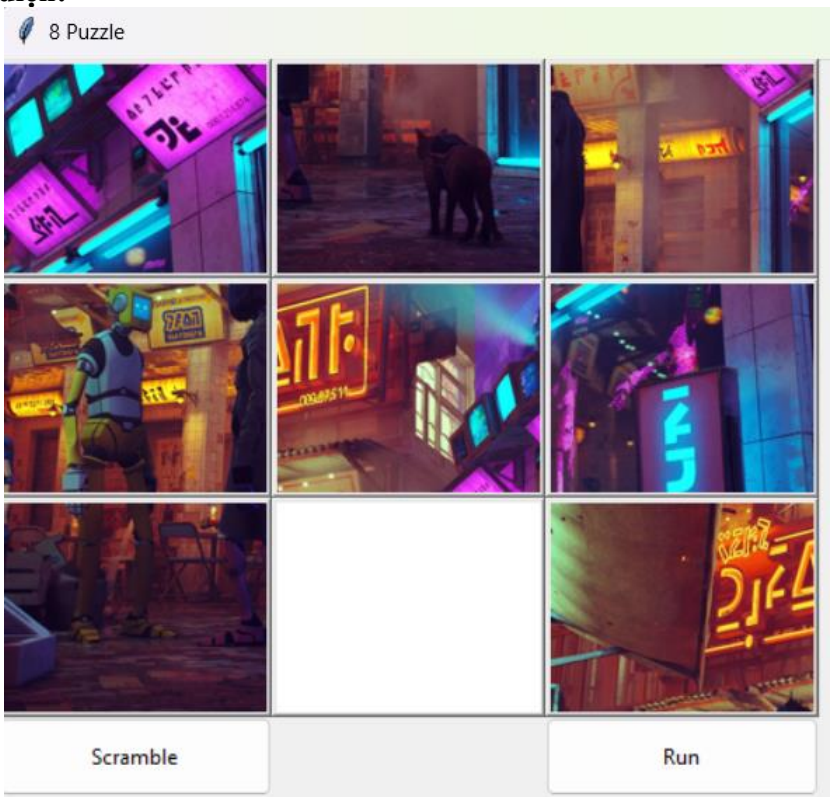
```
import tkinter as tk
from PIL import Image, ImageTk
import cv2
import numpy as np
import os.path
import random
import time
from functools import partial
from tkinter import *
from tkinter import ttk
import tkinter as tk
from PIL import Image, ImageTk

from search import astar_search, EightPuzzle
import sys
sys.path.append(os.path.join(os.path.dirname(__file__), '..'))
state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
puzzle = EightPuzzle(tuple(state))
solution = None

b = [None] * 9
image = cv2.imread('stray-2.jpg')

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
pil_image = Image.fromarray(image)

piece_width = pil_image.width // 3
piece_height = pil_image.height // 3

pieces = []
state_pieces = {}
white = Image.new('RGB', (piece_width, piece_height), (255, 255, 255))
piece_number = 0

for j in range(3):
    for i in range(3):
        left = i * piece_width
        upper = j * piece_height
        right = left + piece_width
        lower = upper + piece_height
        if i==2 and j==2:
            break
        piece = pil_image.crop((left, upper, right, lower))
        pieces.append(piece)
        state_pieces[piece_number] = piece
        piece_number += 1

pieces.append(white)
pieces = pieces[1:] + [pieces[0]]
root = tk.Tk()

root.title('8 Puzzle')

def scramble():
    global state
    global state_pieces
    global pieces
    global puzzle
    possible_actions = ['UP', 'DOWN', 'LEFT', 'RIGHT']
    scramble = []
    for _ in range(60):
        scramble.append(random.choice(possible_actions))
    for move in scramble:
        if move in puzzle.actions(state):
            state = list(puzzle.result(state, move))
            puzzle = EightPuzzle(tuple(state))
            state_pieces = {state[i]: pieces[i] for i in range(9)}
            create_buttons()
    print(state)
    for i, piece in state_pieces.items():
        print(i, piece)
def solve():
    return astar_search(puzzle).solution()
def solve_steps():
```

```
    global puzzle
    global solution
    global state
    global state_pieces
    global pieces
    solution = solve()
    print(solution)
    for move in solution:
        state = list(puzzle.result(state, move))
        puzzle = EightPuzzle(tuple(state))
        state_pieces = {state[i]: pieces[i] for i in range(9)}
        create_buttons()
        root.update()
        root.after(0, time.sleep(0.75))

def exchange(index):
    global state
    global solution
    global puzzle
    global state_pieces
    global pieces
    zero_ix = list(state).index(0)
    actions= puzzle.actions(state)
    current_action = ''

    i_diff = index // 3 - zero_ix // 3
    j_diff = index % 3 - zero_ix % 3
    if i_diff == 1:
        current_action += 'DOWN'
    elif i_diff == -1:
        current_action += 'UP'
    if j_diff == 1:
        current_action += 'RIGHT'
    elif j_diff == -1:
        current_action += 'LEFT'
    if abs(i_diff) + abs(j_diff) != 1:
        current_action = ''
    if current_action in actions:
        b[zero_ix].grid_forget()
        b[zero_ix] = tk.Button(root, image=state_pieces[index], width=150,
command=partial(exchange, zero_ix))
        state_pieces[zero_ix], state_pieces[index] = state_pieces[index], state_pieces[zero_ix]
        puzzle = EightPuzzle(tuple(state_pieces.keys()))


def create_buttons():
    new_size = (150, 150)
    for i, piece in state_pieces.items():
        h = piece.thumbnail(new_size)
        tk_piece = ImageTk.PhotoImage(piece)
```

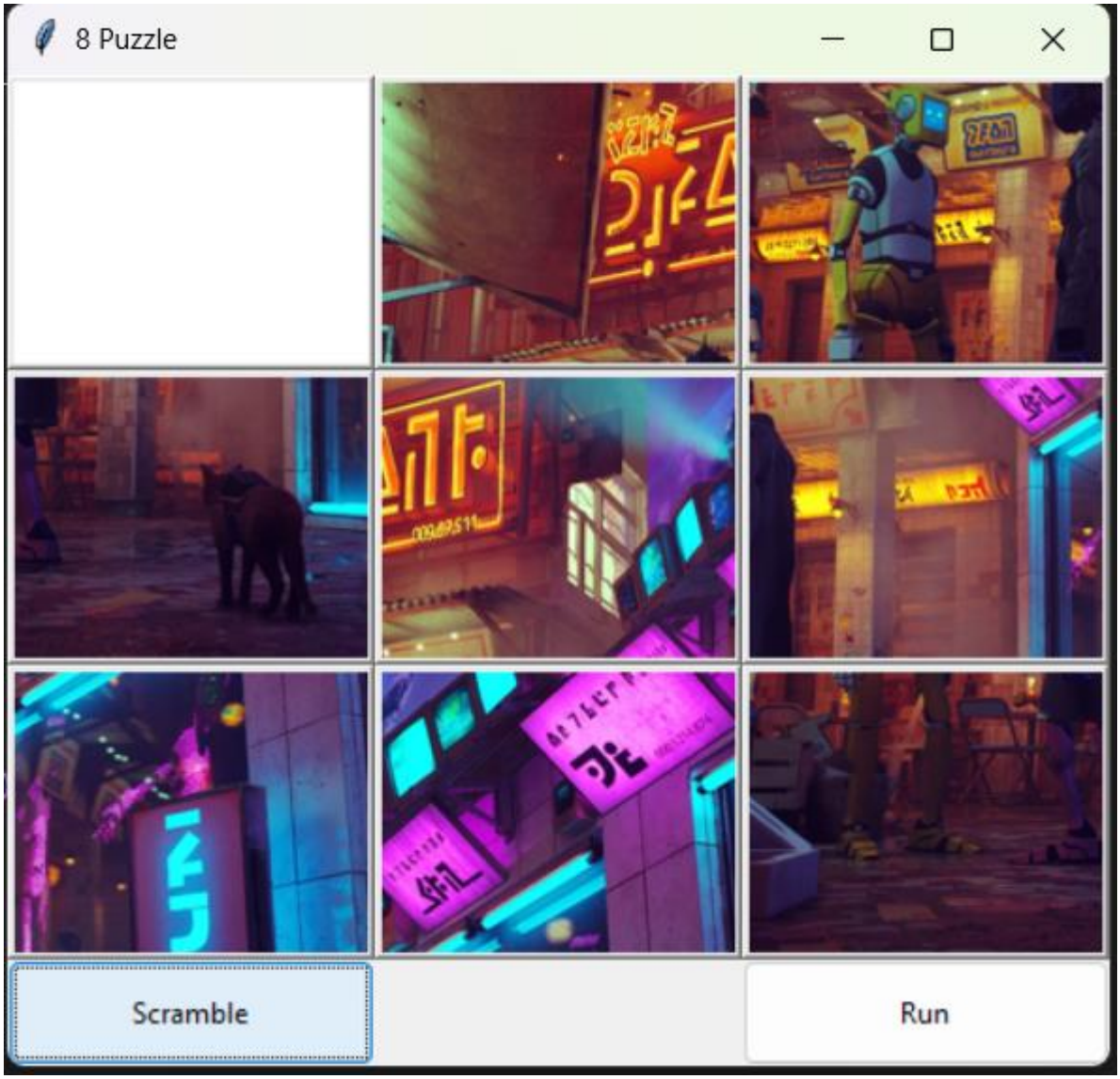
```
        b[i] = tk.Button(root, image=tk_piece, width= h, command=partial(exchange, i))
        b[i].grid(row=i // 3, column=i % 3)
        b[i].image = tk_piece


def create_static_buttons():
    scramble_btn = ttk.Button(root, text='Scramble', width=8, command=partial(scramble))
    scramble_btn.grid(row=3, column=0, ipady=10, sticky=tk.EW)
    run_btn = ttk.Button(root, text='Run', width=8, command=partial(solve_steps))
    run_btn.grid(row=3, column=2, ipady=10, sticky=tk.EW)

def init():
    global state
    global solution
    global puzzle
    global pieces
    global state_pieces
    state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    scramble()
    create_buttons()
    create_static_buttons()
init()
root.mainloop()
```

Kết quả giao diện:



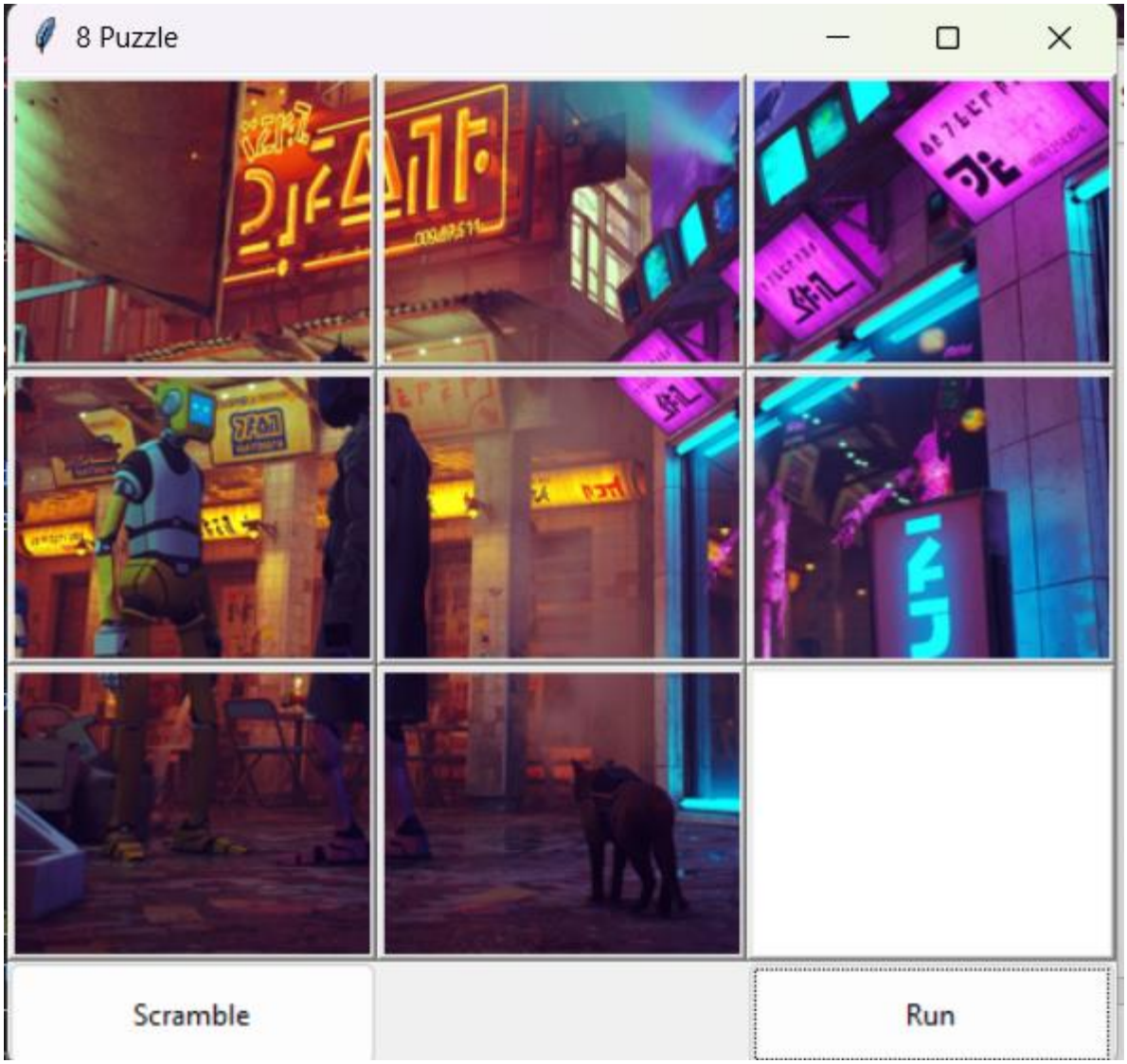
Hình 17 Giao diện GUI của bài toán ghép tranh

Sau khi “Scramble”:



Hình 18. Giao diện GUI của bài toán ghép tranh

Sau khi “Run”:



Hình 19. Giao diện GUI của bài toán ghép tranh

Kết quả in ra màn hình:
'RIGHT', 'UP', 'LEFT', 'UP', 'LEFT', 'DOWN', 'RIGHT', 'DOWN', 'LEFT', 'UP', 'RIGHT', 'UP', 'LEFT', 'DOWN', 'RIGHT', 'DOWN', 'RIGHT', 'UP', 'LEFT', 'UP', 'RIGHT', 'DOWN', 'DOWN'] **[**

KẾT LUẬN

Đề tài đã trình bày cơ sở lý thuyết của thuật toán A*, cách cấu hình cũng như những ứng dụng trong giải quyết bài toán thực tế:

1. Tìm đường đi ngắn nhất từ một thành phố đến thành phố khác với khoảng cách tính theo đường chim bay
2. Tìm đường đi ngắn nhất trong mê cung, đi từ ‘x’ đến ‘o’
3. Giải quyết bài toán 8-Puzzle và ứng dụng để cấu hình thành bài toán xếp mảnh ghép ảnh.

Từ những ứng dụng trên, cho thấy tầm quan trọng của thuật toán A* trong việc tìm kiếm là rất lớn. Và từ đó, rút ra kết luận đặc điểm và nhược điểm của thuật toán A*:

Đặc điểm của thuật toán A*:

1. Thuật toán cho kết quả tối ưu nếu hàm heuristic h là hàm chấp nhận được.

2. Thuật toán đầy đủ, trừ trường hợp có vô số các nút với hàm f có giá trị rất nhỏ nằm giữa node xuất phát và node đích.

3. Độ phức tạp: trong trường hợp xấu nhất, khi hàm heuristic không có nhiều thông tin, độ phức tạp tính toán và yêu cầu bộ nhớ của A* đều là $O(b^m)$.

4. Trong tất cả các thuật toán tìm kiếm tối ưu sử dụng cùng hàm heuristics thì thuật toán A* có độ phức tạp tính toán nhỏ nhất, tức là yêu cầu sinh ra ít nút nhất trước khi tìm ra lời giải.

Nhược điểm của thuật toán A*: Yêu cầu bộ nhớ lớn. Do đó, A* thường không được sử dụng để giải các bài toán có kích thước lớn.

TÀI LIỆU THAM KHẢO

1. Alberto Artasanchez Prateek Joshi, Artificial Intelligence with Python, Packt Publishing Ltd, 2nd Edition, 2020.

2. Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Pearson Education ©, 2021.

3. Từ Minh Phương, Giáo trình Nhập môn trí tuệ nhân tạo, Học Viện Công nghệ Bưu chính Viễn thông, 2014.

4. George F. Luger, William A. Stubblefield – Albuquerque – Artificial Intelligence – Wesley Publishing Company, Inc – 1997.