Code:

from typing import List, Tuple,Set

from collections import deque

import heapq

import copy

import matplotlib

matplotlib.use("Agg")

import matplotlib.pyplot as plt

from matplotlib.patches import Rectangle

import os

import sys

import subprocess

class SearchAlgorithm:

@staticmethod

def get\_neighbors(x: int, y: int, grid: List[List[str]]) -> List[Tuple[int, int]]:

rows, cols = len(grid), len(grid[0])

neighbours = []

directions = [(0,1), (1,0), (0,-1), (-1,0)]

for dx, dy in directions:

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < rows and 0 <= new\_y < cols and grid[new\_x][new\_y] != '-1':

neighbours.append((new\_x, new\_y))

return neighbours

def get\_hueristics(grid: List[List[str]]):

start,target=SearchAlgorithm.get\_start\_target(grid)

x,y=target

if start==-1 or target == -1:

return -1, [],[]

hueristics={}

for row in range(len(grid)):

for col in range(len(grid[0])):

if grid[row][col] != "-1":

hueristics[(row, col)] = abs(x - row) + abs(y - col)

return hueristics

@staticmethod

def get\_start\_target(grid: List[List[str]]) -> Tuple[Tuple[int, int], Tuple[int, int]]:

start, target = None, None

for row in range(len(grid)):

for col in range(len(grid[0])):

if grid[row][col] == "s":

start = (row, col)

elif grid[row][col] == "t":

target = (row, col)

if start is None or target is None:

return -1, grid

return start, target

def best\_first\_search(grid: List[List[str]]) -> Tuple[int, List[Tuple[int, int]]]:

start, target = SearchAlgorithm.get\_start\_target(grid)

cost=1

if start == -1 or target == -1:

return -1, [],[]

hueristics=SearchAlgorithm.get\_hueristics(grid)

priority\_queue = [(hueristics[start], start)]

visited = set()

parents = {}

distances = {start: 0}

Traversal=[]

while priority\_queue:

distance, (x, y) = heapq.heappop(priority\_queue)

if (x, y) == target:

path = []

while (x, y) in parents:

path.append((x, y))

(x, y) = parents[(x, y)]

path.append(start)

path.reverse()

for (x,y) in path:

if grid[x][y]!='s' or grid[x][y]!='t':

grid[x][y]=cost

cost+=1

Traversal.append(target)

return distances[target],path,visited

Traversal.append((x,y))

for neighbor in SearchAlgorithm.get\_neighbors(x, y, grid):

new\_distance=distances[(x,y)]+1

if neighbor not in distances or new\_distance<distances[neighbor]:

heapq.heappush(priority\_queue, (hueristics[neighbor], neighbor))

parents[neighbor] = (x, y)

distances[neighbor] = new\_distance

return -1,[],[]

# Implement A\* Search

def a\_star\_search(grid: List[List[str]]) -> Tuple[int, List[Tuple[int, int]], List[Tuple[int, int]]]:

start, target = SearchAlgorithm.get\_start\_target(grid)

if start == -1 or target == -1:

return -1, [], []

hueristics = SearchAlgorithm.get\_hueristics(grid)

if hueristics == -1:

return -1, [], []

priority\_queue = [(hueristics[start], 0, start)] # (f(n), g(n), node)

expanded = set()

parents = {}

distances = {start: 0}

traversal = []

while priority\_queue:

\_, g, (x, y) = heapq.heappop(priority\_queue)

if (x, y) in expanded:

continue

expanded.add((x, y))

traversal.append((x, y))

if (x, y) == target:

path = []

while (x, y) in parents:

path.append((x, y))

(x, y) = parents[(x, y)]

path.append(start)

path.reverse()

return distances[target], path, expanded

for neighbor in SearchAlgorithm.get\_neighbors(x, y, grid):

new\_distance = g + 1 # Cost from start to neighbor

if neighbor not in distances or new\_distance < distances[neighbor]:

distances[neighbor] = new\_distance

parents[neighbor] = (x, y)

heapq.heappush(priority\_queue, (new\_distance + hueristics[neighbor], new\_distance, neighbor))

return -1, [], []

# Implement Uniform search

def ucs(grid: List[List[str]]) -> Tuple[int, List[Tuple[int, int]], List[Tuple[int, int]]]:

start, target = SearchAlgorithm.get\_start\_target(grid)

if start == -1 or target == -1:

return -1, [], []

priority\_queue = [(0, start)]

expanded = set()

parents = {}

distances = {start: 0}

traversal = []

step\_number = 1

while priority\_queue:

distance, (x, y) = heapq.heappop(priority\_queue)

if (x, y) in expanded:

continue

expanded.add((x, y))

traversal.append((x, y))

if (x, y) == target:

path = []

while (x, y) in parents:

path.append((x, y))

(x, y) = parents[(x, y)]

path.append(start)

path.reverse()

for (rx, ry) in path:

if grid[rx][ry] not in ('s', 't'):

grid[rx][ry] = str(step\_number)

step\_number += 1

return distance, path, expanded

for neighbor in SearchAlgorithm.get\_neighbors(x, y, grid):

if neighbor not in expanded:

new\_cost = distance + 1

old\_cost = distances.get(neighbor, float('inf'))

if new\_cost < old\_cost:

distances[neighbor] = new\_cost

parents[neighbor] = (x, y)

heapq.heappush(priority\_queue, (new\_cost, neighbor))

return -1, [], traversal

def dfs(grid: List[List[str]]) -> Tuple[int, List[List[str]]]:

start,target = SearchAlgorithm.get\_start\_target(grid)

cost=1

if start==-1 or target == -1:

return -1,[],[]

stack=deque([(start,0)])

visited=set([start])

parents={}

distances={start:0}

print

Traversal=[]

while stack:

(x,y),distance=stack.pop()

if (x,y)==target:

path=[]

while (x,y) in parents:

path.append((x,y))

(x,y)=parents[(x,y)]

path.append(start)

path.reverse()

for (x,y) in path:

if grid[x][y]!='s' or grid[x][y]!='t':

grid[x][y]=cost

cost+=1

Traversal.append(target)

return distance,path,visited

Traversal.append((x,y))

for neighbor in SearchAlgorithm.get\_neighbors(x,y,grid):

if neighbor not in visited:

stack.append((neighbor,distance+1))

visited.add(neighbor)

parents[neighbor]=(x,y)

return -1,[],[]

def visualize\_grid(grid: List[List[str]], path: List[Tuple[int, int]], visited\_nodes: Set[Tuple[int, int]], output\_file: str):

if not path:

print(f"No valid path found for {output\_file}, skipping visualization.")

return

grid\_copy = [row.copy() for row in grid]

for (x, y) in path:

if grid\_copy[x][y] not in ('s', 't'):

grid\_copy[x][y] = '\*'

rows = len(grid\_copy)

cols = len(grid\_copy[0])

fig, ax = plt.subplots(figsize=(cols \* 1.2, rows \* 1.2))

ax.set\_xlim(0, cols)

ax.set\_ylim(0, rows)

ax.invert\_yaxis()

ax.axis('off')

for i in range(rows):

for j in range(cols):

cell = grid\_copy[i][j]

if cell == 's':

facecolor = "green"

elif cell == 't':

facecolor = "red"

elif cell == '-1':

facecolor = "lightgray"

else:

facecolor = "white"

# Draw the grid cell

rect = Rectangle((j, i), 1, 1, facecolor=facecolor, edgecolor="black")

ax.add\_patch(rect)

ax.text(j + 0.5, i + 0.5, cell, ha="center", va="center", fontsize=14)

if path:

line\_x = [col + 0.5 for (row, col) in path]

line\_y = [row + 0.5 for (row, col) in path]

ax.plot(line\_x, line\_y, color="red", linewidth=3, marker="o", markersize=5)

plt.savefig(output\_file, bbox\_inches='tight')

plt.close(fig)

print(f"Final grid image saved to {output\_file}")

@staticmethod

def bfs(grid: List[List[str]]) -> Tuple[int, List[Tuple[int, int]]]:

start, target = SearchAlgorithm.get\_start\_target(grid)

cost=1

if start == -1 or target == -1:

return -1, [],[]

queue = deque([(start, 0)])

visited = set([start])

parents = {}

distances = {start: 0}

Traversal=[]

while queue:

(x, y), distance = queue.popleft()

if (x, y) == target:

path = []

while (x, y) in parents:

path.append((x, y))

(x, y) = parents[(x, y)]

path.append(start)

path.reverse()

for (x,y) in path:

if grid[x][y]!='s' or grid[x][y]!='t':

grid[x][y]=cost

cost+=1

Traversal.append(target)

return distance,path,Traversal

Traversal.append((x,y))

for neighbor in SearchAlgorithm.get\_neighbors(x, y, grid):

if neighbor not in visited:

queue.append((neighbor, distance + 1))

visited.add(neighbor)

parents[neighbor] = (x, y)

return -1,[],[]

if \_\_name\_\_ == "\_\_main\_\_":

example1 = [

['0', '0', '0', '0'],

['0', '-1', '-1', 't'],

['s', '0', '-1', '0'],

['0', '0', '0', '-1']

]

example2 = [

['0', '0', '0', '0'],

['0', '-1', '-1', 't'],

['s', '0', '-1', '0'],

['0', '0', '0', '-1']

]

example3 = [

['0', '0', '0', '0'],

['0', '-1', '-1', 't'],

['s', '0', '-1', '0'],

['0', '0', '0', '-1']

]

example4 = [

['0', '0', '0', '0'],

['0', '-1', '-1', 't'],

['s', '0', '-1', '0'],

['0', '0', '0', '-1']

]

example5 = [

['0', '0', '0', '0'],

['0', '-1', '-1', 't'],

['s', '0', '-1', '0'],

['0', '0', '0', '-1']

]

print("BFS Traversal")

found, path, Traversal = SearchAlgorithm.bfs(example1)

if found == -1:

print("No path found")

else:

print("Shortest Distance:", found)

print("Shortest Path:", path)

SearchAlgorithm.visualize\_grid(example1, path, Traversal, "bfs\_output.png")

print("Traversal:", Traversal)

print("DFS Traversal")

found, path, Traversal = SearchAlgorithm.dfs(example2)

if found == -1:

print("No path found")

else:

print("Shortest Distance:", found)

print("Shortest Path:", path)

SearchAlgorithm.visualize\_grid(example2, path, Traversal, "dfs\_output.png")

print("Traversal:", Traversal)

print("UCS Traversal")

found, path, Traversal = SearchAlgorithm.ucs(example3)

if found == -1:

print("No path found")

else:

print("Shortest Distance:", found)

print("Shortest Path:", path)

SearchAlgorithm.visualize\_grid(example1, path, Traversal, "ucs\_output.png")

print("Traversal:", Traversal)

print("Best First Traversal")

found, path, Traversal = SearchAlgorithm.best\_first\_search(example4)

if found == -1:

print("No path found")

else:

print("Shortest Distance:", found)

print("Shortest Path:", path)

SearchAlgorithm.visualize\_grid(example4, path, Traversal, "BestFirstSearch\_output.png")

print("Traversal:", Traversal)

print("A\* Search")

found, path, Traversal = SearchAlgorithm.a\_star\_search(example5)

if found == -1:

print("No path found")

else:

print("Shortest Distance:", found)

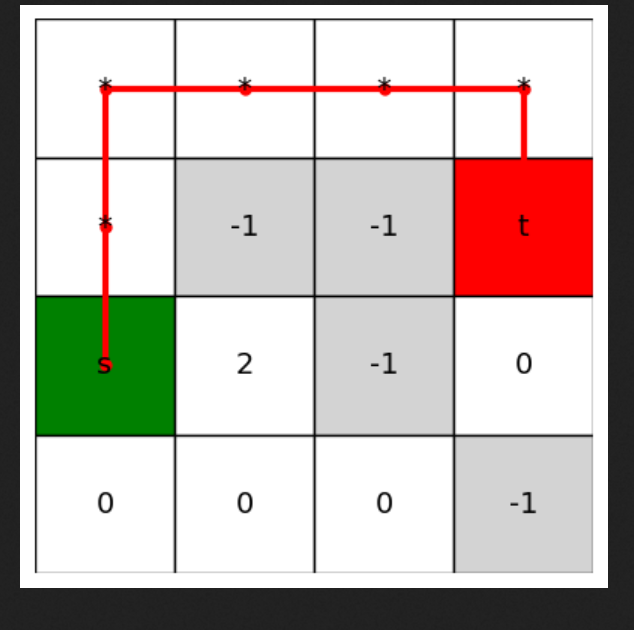
print("Shortest Path:", path)

print("Traversal:", Traversal)

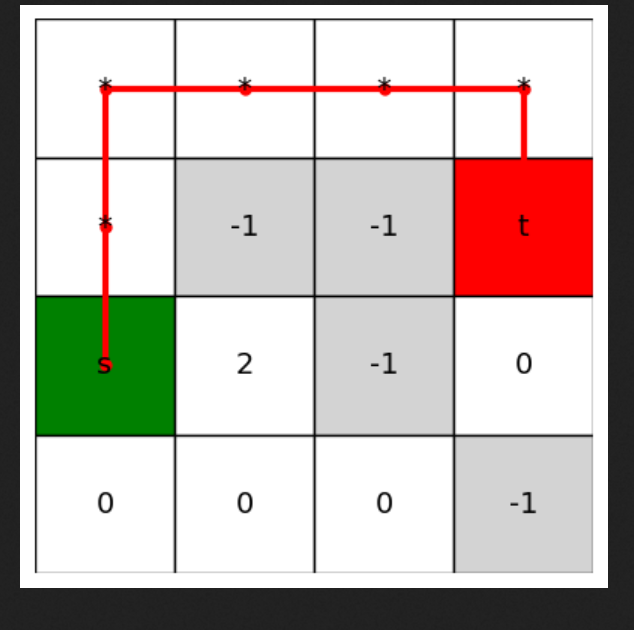
SearchAlgorithm.visualize\_grid(example5, path, Traversal, "A\_star\_output.png")

Outputs:

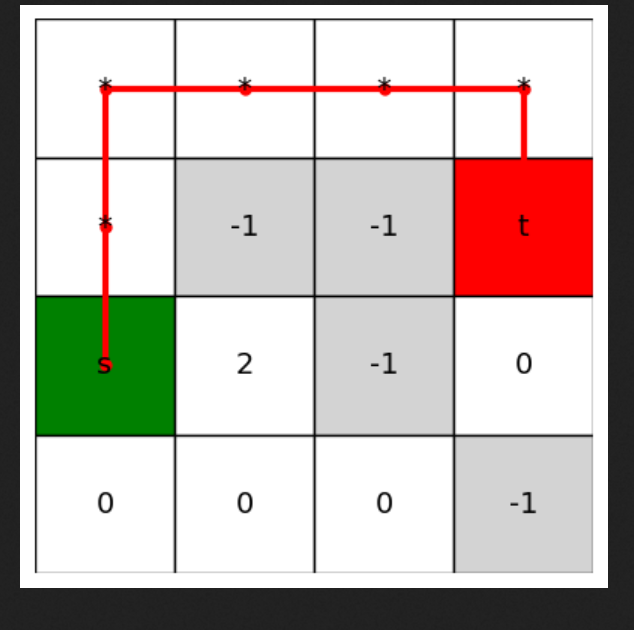
Bfs:

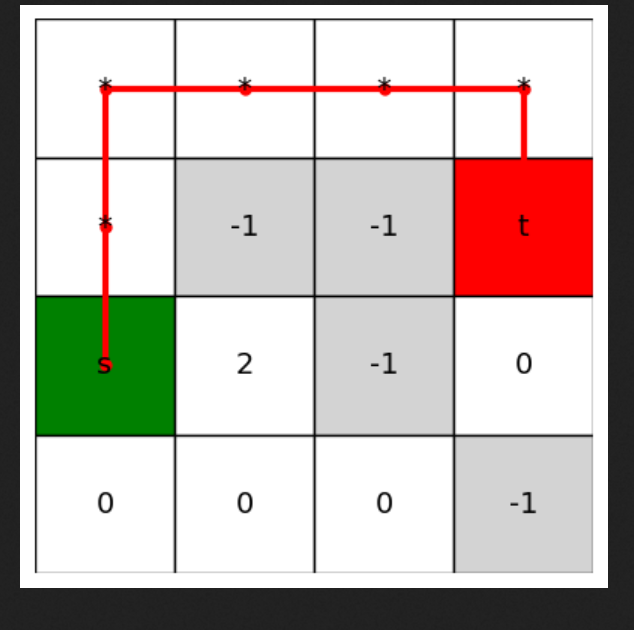


Dfs:



Ucs:

  
  
  
a\*

  
  
best first:  
