As the code snippet shows, the **GBFS** class is a subclass of **BestSearchStrategy** and implements the **search** method that takes a graph **g**, a source vertex **src**, and a destination vertex **dst** as input arguments.

The goal of the **search** method is to perform a Greedy Best-First Search (GBFS) on the given graph **g** starting from the source vertex **src** and stopping when the destination vertex **dst** is found. The method should return two values: a list of the vertices expanded in the traversal order, and the path from the source vertex to the destination vertex.

To implement this method, the following steps need to be performed:

1. Create an empty list called **frontier** to store the nodes to be explored, and add the source vertex **src** to the frontier.
2. Create an empty list called **expanded** to store the nodes that have been expanded, and add the source vertex **src** to the expanded list.
3. Create a dictionary called **came\_from** to keep track of the previous node that leads to the current node, initialized to None for all nodes except the source vertex, which is set to -1.
4. Create a dictionary called **distances** to keep track of the distances from the source vertex to each vertex, initialized to infinity for all vertices except the source vertex, which is set to 0.
5. Create a while loop that runs as long as the frontier is not empty. Inside the loop, do the following:

a. Get the vertex with the lowest heuristic value from the frontier using a priority queue. If the vertex is the destination vertex **dst**, break out of the loop.

b. For each neighbor of the current vertex, check if the neighbor has not been visited yet (i.e., not in the expanded list). If the neighbor is not in the frontier, add it to the frontier and update its distance and **came\_from** values. If the neighbor is already in the frontier, update its distance and **came\_from** values if the new distance is smaller than the current distance.

c. Add the current vertex to the expanded list.

1. After the loop, if the destination vertex **dst** is not in the **came\_from** dictionary, it means that there is no path from the source to the destination, so return the expanded list and an empty path.
2. Otherwise, construct the path from the source to the destination by backtracking from the destination using the **came\_from** dictionary. Add each vertex to the front of the path list.
3. Return the expanded list and the path list as a tuple.

dịch lại đoạn mã

2 / 2

Đoạn mã trên là lớp **GBFS** kế thừa từ lớp **BestSearchStrategy** và triển khai phương thức **search** nhận vào đồ thị **g**, đỉnh nguồn **src** và đỉnh đích **dst** làm tham số đầu vào.

Mục đích của phương thức **search** là thực hiện tìm kiếm theo thuật toán Greedy Best-First Search (GBFS) trên đồ thị **g** bắt đầu từ đỉnh nguồn **src** và dừng lại khi tìm thấy đỉnh đích **dst**. Phương thức này sẽ trả về hai giá trị: một danh sách các đỉnh đã được khai triển theo thứ tự và đường đi từ đỉnh nguồn đến đỉnh đích.

Để triển khai phương thức này, cần thực hiện các bước sau:

1. Tạo một danh sách rỗng có tên là **frontier** để lưu trữ các nút để khám phá và thêm đỉnh nguồn **src** vào **frontier**.
2. Tạo một danh sách rỗng có tên là **expanded** để lưu trữ các nút đã được khai triển và thêm đỉnh nguồn **src** vào **expanded**.
3. Tạo một từ điển có tên là **came\_from** để theo dõi đỉnh trước đó dẫn đến đỉnh hiện tại, được khởi tạo là None cho tất cả các đỉnh ngoại trừ đỉnh nguồn, được đặt là -1.
4. Tạo một từ điển có tên là **distances** để theo dõi khoảng cách từ đỉnh nguồn đến mỗi đỉnh, được khởi tạo là vô cùng cho tất cả các đỉnh ngoại trừ đỉnh nguồn, được đặt là 0.
5. Tạo một vòng lặp while chạy cho đến khi **frontier** không còn trống nữa. Bên trong vòng lặp, thực hiện các bước sau:

a. Lấy đỉnh có giá trị heuristic thấp nhất từ **frontier** sử dụng hàng đợi ưu tiên. Nếu đỉnh đó là đỉnh đích **dst**, thoát khỏi vòng lặp.

b. Đối với mỗi đỉnh kề của đỉnh hiện tại, kiểm tra xem đỉnh kề đó chưa được ghé thăm (tức không nằm trong danh sách **expanded**). Nếu đỉnh kề đó chưa có trong **frontier**, thêm nó vào **frontier** và cập nhật giá trị khoảng cách và **came\_from** của nó. Nếu đỉnh kề đó đã có trong **frontier**, cập nhật

from queue import PriorityQueue

from typing import List, Tuple

from graph import Graph

from best\_search\_strategy import BestSearchStrategy

class GBFS(BestSearchStrategy):

def search(self, g: Graph, src: int, dst: int) -> Tuple[List[int], List[int]]:

# create an empty list to store nodes to be explored

frontier = PriorityQueue()

# add the source vertex to the frontier with priority based on heuristic value

frontier.put((g.heuristic(src, dst), src))

# create an empty list to store expanded nodes in the traversal order

expanded = []

# create a dictionary to keep track of the previous node on the path

came\_from = {v: -1 for v in g.vertices()}

# set the previous node of the source vertex to itself

came\_from[src] = src

# create a dictionary to keep track of the distance from the source node to each node

distances = {v: float('inf') for v in g.vertices()}

# set the distance of the source vertex to 0

distances[src] = 0

while not frontier.empty():

# get the node with the lowest heuristic value from the frontier

current = frontier.get()[1]

# add the current node to the expanded list

expanded.append(current)

# if the current node is the destination node, stop the search

if current == dst:

break

# explore the neighbors of the current node

for neighbor in g.neighbors(current):

# calculate the distance from the source node to the neighbor node

new\_distance = distances[current] + g.get\_edge\_weight(current, neighbor)

# if the neighbor node has not been visited or the new distance is smaller than the current distance

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

# update the distance and previous node of the neighbor node

distances[neighbor] = new\_distance

came\_from[neighbor] = current

# add the neighbor node to the frontier with priority based on heuristic value

frontier.put((g.heuristic(neighbor, dst), neighbor))

# create a list to store the path from the source node to the destination node

path = []

# if a path was found, construct the path by backtracking from the destination node

if came\_from[dst] != -1:

node = dst

while node != src:

path.append(node)

node = came\_from[node]

path.append(src)

path.reverse()

return expanded, path

Lớp GBFS này sử dụng một hàng đợi ưu tiên để lưu trữ các đỉnh cần khám phá, với ưu tiên được xác định dựa trên giá trị heuristic của mỗi đỉnh. Các đỉnh đã được khai triển được lưu trữ trong danh sách expanded. Từ điển came\_from được sử dụng để theo dõi đỉnh trước đó trên đường đi từ đỉnh nguồn đến đỉnh đích, và từ điển distances được s