BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

----------※※※----------

BÁO CÁO

Cơ Sở Trí Tuệ Nhân Tạo

Lab 01
SEARCH STRATEGIES

Giảng viên lý thuyết: Bùi Tiến Lên

Giảng viên hướng dẫn thực hành: Trần Quốc Huy, Phạm Trọng Nghĩa

Thành phố Hồ Chí Minh - 2023

MỤC LỤC

I. Thông tin sinh viên:

Tên	MSSV	Lớp
TÔ KHÁNH LINH	21127638	21CLC08

II. Đánh giá mức độ hoàn thành:

Phần	Mô tả	Hoàn thành
1	Breadth - first search(BFS)	**100%**
2	Depth first Search (DFS)	**100%**
3	Uniform-cost search (UCS)	**100%**
4	Greedy Best First Search	**100%**
5	Graph- search A*	**100%**

Mức độ hoàn thành: 100%

III. Cơ chế của hàm:

1. Cấu trúc file input:

- File lưu thông tin dưới dạng danh sách kề.
- Dòng đầu là size, dòng thứ hai là lối vào và lối ra.
- Với mỗi dòng cạnh được lưu trước sau đó mới tới cạnh kề của cạnh đó. Ví dụ: 64 32 35.Thì 32 và 35 kề với 64.

2. Cấu trúc hàm readfile(string fileread):

readfile(fileread)

- **Trả về:** maze(graph), begin (cổng), end (đích), n (kích thước mê cung)
- Sau khi đọc hết các cạnh kề vào list1
- Ta chuyển list1 (list) sang maze (dictionary) bằng cách dùng vòng lặp for với x là từng phần tử của list1.
- Maze sẽ có key là phần tử đầu của từng phần tử list1 và giá trị của key đó là phần tử tiếp theo sau phần tử đầu tiên của list: maze[x[0]]=x[1:]

3. Cấu trúc hàm DFS(Depth first Search):

DFS(maze,begin,end)

- **Trả về:** result(Đường ra khỏi mê cung), visited (Những cạnh ghé thăm), elapsed_time (Thời gian chạy của hàm)
- Tạo kiểu dữ liệu list cho stack, path,visited và dictionary cho path1{}. Bỏ phần tử đầu tiên (lối vào) vào stack.
- Path1{} dùng để lưu lại các đỉnh đã đi qua các cạnh

- Lặp đến khi không còn gì trong stack, **lấy phần tử trên cùng trong stack.** Đó là phần tử ta đang duyệt nên ta bỏ nó vào visited tránh sử dụng thêm lần nữa.
- Nếu phần tử (v) không bằng lối ra thì ta duyệt các cạnh kề của phần tử đó với vòng lặp for. Kiểm tra các cạnh kề có trong visited không, nếu không bỏ các phần tử (v) đó vào stack, path1 lưu lại các cạnh kề liên kết với cạnh hiện tại.
- Nếu phần tử (v) bằng với lối ra (end) thì, ta lấy path (đường đi ra khỏi mê cung) bằng cách cho một vòng lặp while đến khi phần tử (v) bằng lối vào (begin) thì dừng. Tìm cạnh nối với cạnh hiện tại bằng path1. Vòng lặp dừng thì ta thêm lối vào ở cuối path. Và đảo ngược path vì ta lấy từ cuối đến đầu.

4. <u>Cấu trúc hàm BFS (Breadth - first search):</u>

BFS(maze,begin,end)

- **Trả về:** result(Đường ra khỏi mê cung), visited (Những cạnh ghé thăm), elapsed_time (Thời gian chạy của hàm)
- Tạo kiểu dữ liệu list cho stack, path,visited và dictionary cho path1{}. Kiểu dữ liệu queue hai đầu (deque). Thêm phần tử đầu tiên (lối vào) vào queue
- Path1{} dùng để lưu lại các đỉnh đã đi qua các cạnh
- Lặp đến khi không còn gì trong queue, **lấy phần tử đầu tiên của hàng đợi.** Đó là phần tử ta đang

duyệt nên ta bỏ nó vào visited tránh sử dụng thêm lần nữa.

- Nếu phần tử (v) không bằng lối ra (end) thì ta duyệt các cạnh kề (neighbor) của phần tử đó với vòng lặp for. Kiểm tra các cạnh kề (neighbor) có trong visited không, nếu không bỏ các phần tử (neighbor)đó vào queue, path1 lưu lại các cạnh kề liên kết với cạnh hiện tại.
- Nếu phần tử (v) bằng với lối ra (end) thì, ta lấy path (đường đi ra khỏi mê cung) bằng cách cho một vòng lặp while đến khi phần tử (v) bằng lối vào (begin) thì dừng. Tìm cạnh nối (neighbor)với cạnh hiện tại (v) bằng path1. Vòng lặp dừng thì ta thêm lối vào(begin) ở cuối path. Và đảo ngược path vì ta lấy từ cuối đến đầu.

5. <u>Cấu trúc hàm UCS (Uniform-cost search):</u>

UCS(maze,begin,end)

- **Trả về:** result(Đường ra khỏi mê cung), visited (Những cạnh ghé thăm), elapsed_time (Thời gian chạy của hàm)
- Tạo kiểu dữ liệu list cho stack, path,visited và dictionary cho path1{}. Kiểu dữ liệu queue (PriorityQueue). Thêm phần tử đầu tiên (lối vào) vào queue với cost=0.
- Path1{} dùng để lưu lại các đỉnh đã đi qua các cạnh.
- Lặp đến khi không còn gì trong queue, **lấy phần tử đầu tiên của hàng đợi (phần tử có cost nhỏ**

nhất). Đó là phần tử ta đang duyệt nên ta bỏ nó vào visited tránh sử dụng thêm lần nữa.

- Nếu cạnh đang duyệt (v) không bằng lối ra (end) thì ta duyệt các cạnh kề (neighbor) của cạnh đó (v) với vòng lặp for. Tính cost của từng cạnh (neighbor) bằng cách cộng 1 vào cost cạnh đang duyệt. Kiểm tra các cạnh kề (neighbor) có trong visited không, nếu không bỏ các phần tử đó (neighbor) vào queue cùng cost của cạnh đó, path1 lưu lại các cạnh kề liên kết với cạnh hiện tại.
- Nếu phần tử (v) bằng với lối ra (end) thì, ta lấy path (đường đi ra khỏi mê cung) bằng cách cho một vòng lặp while đến khi phần tử (v) bằng lối vào (begin) thì dừng. Tìm cạnh nối với cạnh hiện tại bằng path1. Vòng lặp dừng thì ta thêm lối vào ở cuối path. Và đảo ngược path vì ta lấy từ cuối đến đầu.

6. **<u>Cấu trúc hàm manhattan_distance (Tính khoảng cách Manhattan)</u>**

manhattan_distance(begin, end, n)

- **Trả về:** Khoảng cách manhattan
- Lấy begin(điểm bắt đầu) và end (đích) chia dư cho size(kích thước mê cung) ta sẽ có được tọa độ của 2 điểm đó. Sau đó áp dụng công thức manhattan đã sẽ có được khoảng cách.

7. Cấu trúc hàm GDFS (Greedy Best First Search):

GDFS(maze,begin,end,sizemaze)

- **Trả về:** result(Đường ra khỏi mê cung), visited (Những cạnh ghé thăm), elapsed_time (Thời gian chạy của hàm)
- Tạo kiểu dữ liệu list cho stack, path,visited và dictionary cho path1{}. Kiểu dữ liệu queue (PriorityQueue). Thêm phần tử đầu tiên (lối vào) vào queue với cost là khoảng cách manhattan từ lối vào đến đích.
- Path1{} dùng để lưu lại các đỉnh đã đi qua các cạnh.
- Lặp đến khi không còn gì trong queue, **lấy phần tử đầu tiên của hàng đợi (phần tử có cost nhỏ nhất).** Đó là phần tử ta đang duyệt nên ta bỏ nó vào visited tránh sử dụng thêm lần nữa.
- Nếu cạnh đang duyệt (v) không bằng lối ra (end) thì ta duyệt các cạnh kề(neighbor) của cạnh đó (v) với vòng lặp for. Tính cost của từng cạnh (neighbor) bằng cách tính khoảng cách manhattan của cạnh đến đích. Kiểm tra các cạnh kề có trong visited không, nếu không bỏ các cạnh kề (neighbor) đó vào queue cùng cost của cạnh đó (neighbor), path1 lưu lại các các cạnh kề (neighbo) này liên kết với cạnh hiện tại (v).

- Nếu phần tử (v) bằng với lối ra (end) thì, ta lấy path (đường đi ra khỏi mê cung) bằng cách cho một vòng lặp while đến khi phần tử (v) bằng lối vào (begin) thì dừng. Tìm cạnh nối với cạnh hiện tại bằng path1. Vòng lặp dừng thì ta thêm lối vào ở cuối path. Và đảo ngược path vì ta lấy từ cuối đến đầu.

8. Cấu trúc hàm astar(Graph- search A*) :

astar(maze,begin,end,sizemaze)

- **Trả về:** result(Đường ra khỏi mê cung), visited (Những cạnh ghé thăm), elapsed_time (Thời gian chạy của hàm)
- Tạo kiểu dữ liệu list cho stack, path,visited và dictionary cho path1{}. Kiểu dữ liệu queue (PriorityQueue). Thêm phần tử đầu tiên (lối vào) vào queue với cost là khoảng cách manhattan từ lối vào đến đích.
- Path1{} dùng để lưu lại các đỉnh đã đi qua các cạnh.
- Lặp đến khi không còn gì trong queue, lấy phần tử đầu tiên của hàng đợi (phần tử có cost nhỏ nhất). Đó là phần tử ta đang duyệt nên ta bỏ nó vào visited tránh sử dụng thêm lần nữa.
- Nếu cạnh đang duyệt (v) không bằng lối ra (end) thì ta duyệt các cạnh kề (neighbor) của cạnh đó (v) với vòng lặp for. Tính cost của từng cạnh (Khoảng cách manhattan của cạnh đến đích cộng với cost cạnh trước cộng 1). Kiểm tra các cạnh kề (neighbor) có trong visited không, nếu không bỏ

các cạnh kề đó (neighbor) vào queue cùng cost của cạnh đó, path1 lưu lại các các cạnh kề (neighbor) này liên kết với cạnh hiện tại (v).

- Nếu phần tử (v) bằng với lối ra (end) thì, ta lấy path (đường đi ra khỏi mê cung) bằng cách cho một vòng lặp while đến khi phần tử (v) bằng lối vào (begin) thì dừng. Tìm cạnh nối với cạnh hiện tại bằng path1. Vòng lặp dừng thì ta thêm lối vào ở cuối path. Và đảo ngược path vì ta lấy từ cuối đến đầu.

9. <u>Cách sử dụng hàm main:</u>

- Cần truyền tên file input và output:
  + Input: File text (mê cung dưới dạng danh sách kề).
  + Output: File text.

IV. <u>Các nguồn tài liệu tham khảo:</u>

- Slide của thầy Bùi Tiến Lên.
- https://www.baeldung.com/cs/dfs-vs-bfs-vs-dijkstra.
- https://www.baeldung.com/cs/find-path-uniform-cost-search