TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA TP. HCM

Cơ sở trí tuệ nhân tạo

Đồ án: Ứng dụng các thuật toán tìm kiếm

19120615 – Hùng Ngọc Phát
19120621 – Lê Minh Phục

Ngày 11 tháng 11 năm 2021

Mục lục

1 Giới thiệu **2**
1.1 Về thành viên nhóm và phân công 2
1.2 Về kiến trúc của chương trình, kiến trúc dữ liệu và processing pipeline 2
1.3 Kiến trúc chương trình 2
1.4 Quá trình tiền xử lý mê cung 3

2 Tài liệu tham khảo **5**

Chương 1

Giới thiệu

1.1 Về thành viên nhóm và phân công

Nhóm có 2 thành viên

Mức độ hoàn thành của từng hạng mục

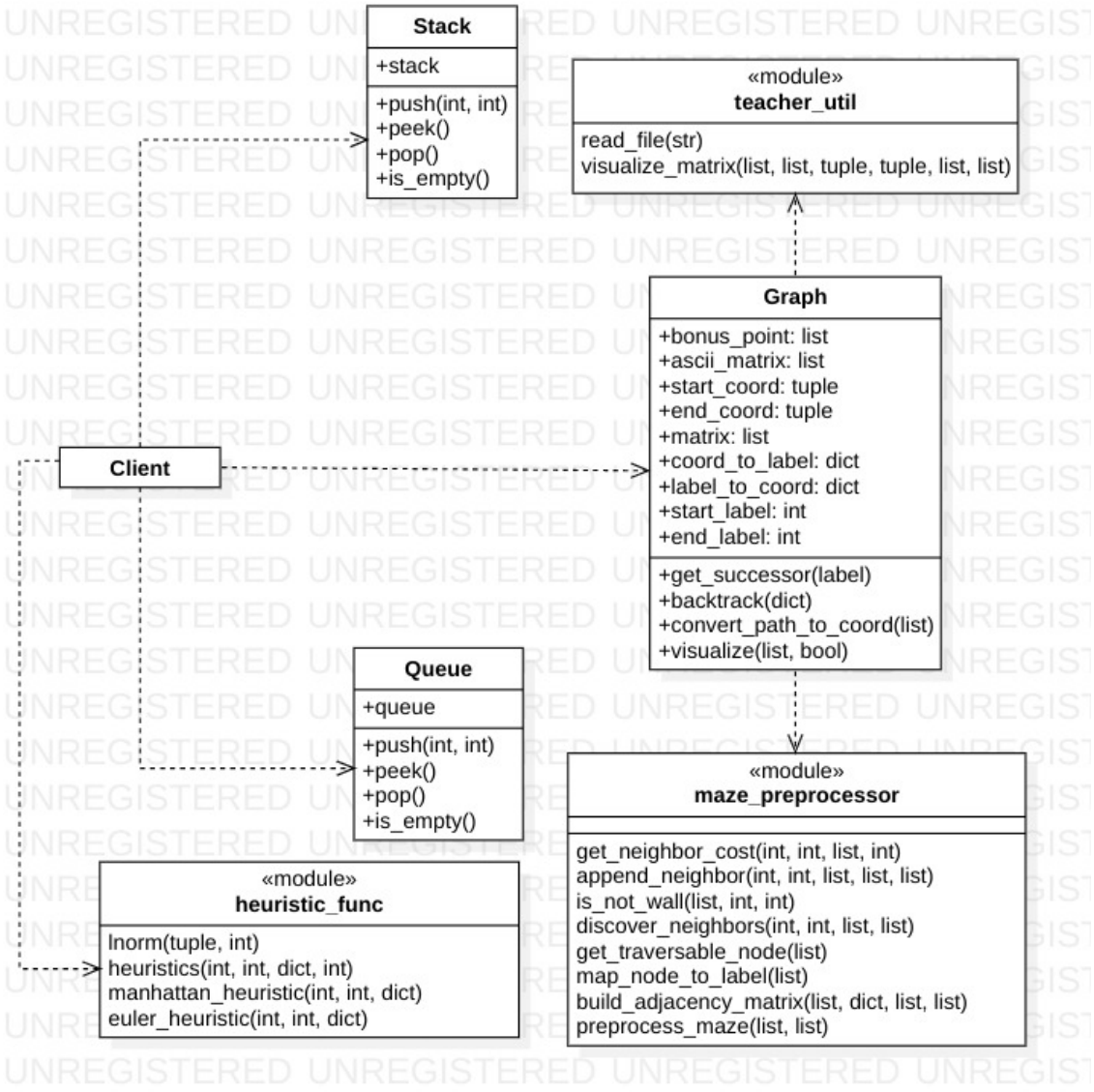
Hạng mục	Không điểm thưởng	Có điểm thưởng
GBFS	100%	-
DFS	100%	-
BFS		
A*		

1.2 Về kiến trúc của chương trình, kiến trúc dữ liệu và processing pipeline

1.3 Kiến trúc chương trình

"Chương trình báo cáo" này được viết bằng Jupyter Notebook để dễ dàng visualize các hình ảnh cũng như tiện lợi cho việc tính toán, sau đó được xuất sang LaTeX. Thầy có thể thử chạy lại notebook báo cáo này trong thư mục `report-src`.

Chương trình gồm các lớp đối tượng sau:



Ghi chú: các "class" có gắn `<<module>>` không phải là class mà chỉ là tập hợp các hàm trong 1 file `*.py` (module).

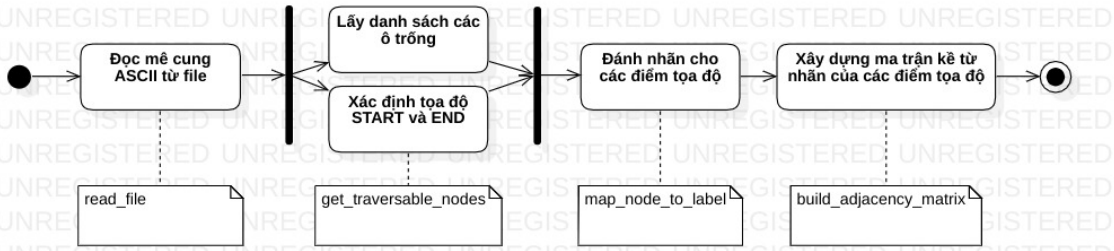
- `Stack`: lớp đối tượng ngăn xếp.
- `Queue`: lớp đối tượng cho hàng đợi và hàng đợi ưu tiên.
- `Graph`: lớp đối tượng cho một đồ thị được sinh ra từ mê cung.
- `maze_preprocessor`: module chứa các hàm tiền xử lý mê cung.
- `heuristic_func`: module chứa các hàm heuristic.
- `teacher_util`: module chứa 2 hàm mà thầy cung cấp trong notebook.

Mỗi hàm nhóm em viết đều có một docstring mô tả input/output và chức năng. Thầy có thể tham khảo source code để biết thêm.

1.4 Quá trình tiền xử lý mê cung

- Để các thuật toán tìm kiếm đường đi có thể chạy được, ta cần chuyển đổi mê cung dưới dạng ASCII như trong file `txt` thành một đồ thị. Ở đây nhóm em đã quyết định chọn một ma trận kề để biểu diễn một đồ thị sinh ra từ mê cung.

- Quá trình tiền xử lý mê cung được thực hiện bởi các hàm trong module `maze_preprocessor`, theo thứ tự sau:



1. **Đọc mê cung từ file ASCII:** sử dụng hàm thầy đã cung cấp, đọc lên mê cung dưới dạng ASCII và danh sách các điểm thưởng.
2. **Lấy danh sách các ô trống & xác định tọa độ START & END**: quét mê cung từ trên xuống dưới, từ trái sang phải để xác định các ô mà tác nhân "có thể đi được" (tức tất cả các ô mà không phải tường). Song song với đó trong lúc quét cũng xác định tọa độ của điểm START và END.
3. **Đánh nhãn cho các điểm tọa độ**: vì mỗi ô đi được trên mê cung được biểu diễn bằng một tọa độ (1 cặp số). Vị chi mỗi cạnh của đồ thị cần 2 cặp số mới biểu diễn được. Tuy nhiên trên ma trận kề chỉ có thể sử dụng 1 cặp số để đánh dấu 1 cạnh. Đó là lý do nhóm em đã quyết định đánh số mỗi điểm tọa độ đi được thành 1 nhãn nguyên để thuận tiện hơn trong việc thiết lập ma trận kề. Sau khi tìm được đường đi trên ma trận kề có thể truy ngược lại tọa độ của chúng để truyền vào hàm `visualize_matrix` mà thầy đã cung cấp. Ví dụ như:

4. **Thiết lập ma trận kề**: quét các ô đi được đã tìm được ở trên và tìm các ô kề với nó, xây dựng nên một hữu hướng đơn đồ thị (có dạng lưới). Từ đó xây dựng được ma trận kề của đồ thị này. Có nghĩa là, đồ thị được sinh ra này có các đỉnh là các ô đi được trên mê cung (không chứa các ô tường).

Về trọng số các cạnh: cạnh nối 2 đỉnh "thường" (không có đỉnh nào là điểm thưởng) sẽ có trọng số mặc định là 50 (ở cả 2 chiều). Còn cạnh nối từ đỉnh "thường" vào đỉnh "điểm thưởng" sẽ có trọng số là 50 + mức điểm thưởng (điểm thưởng < 0, theo như file mẫu). Bằng cách này, các thuật toán mà có "để ý" đến trọng số nối giữa các cạnh (như A*) sẽ có thể một phần nào đó "ăn" được các điểm thưởng này nếu số lượng đủ nhiều và nằm rải rác trên đường đi từ START đến END.

In [ ]:

Chương 2

Tài liệu tham khảo

1. Slide bài giảng và lecture note trên Moodle.
2. From an ASCII Maze to an Adjacency List, https://mazes.readthedocs.io/en/latest/middle.html