TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
Khoa: Công nghệ thông tin

BÁO CÁO ĐỒ ÁN MÔN HỌC
CƠ SỞ TRÍ TUỆ NHÂN TẠO

Lab 01: Các thuật toán tìm kiếm

Giảng viên hướng dẫn:
Nguyễn Duy Khánh
Nguyễn Ngọc Băng Tâm

TP Hồ Chí Minh, ngày 22 tháng 10 năm 2022

MỤC LỤC

I. Thành viên nhóm:

STT	Họ và tên	Mã số sinh viên
1	Phạm Võ Hải Đăng	20120263
2	Trương Cao Hoàng Gia	20120279
3	Võ Minh Hiếu	20120289

II. Phân công công việc:

Thành viên	Công việc
Phạm Võ Hải Đăng	Thiết kế và cài đặt thuật toán BFS, DFS cho bản đồ không điểm thưởng.
Trương Cao Hoàng Gia	Thiết kế và cài đặt thuật toán GBFS, A* cho bản đồ không điểm thưởng.
Võ Minh Hiếu	Thiết kế và cài đặt thuật toán UCS, thiết kế mê cung.

III. Đánh giá thuật toán:

1. Thuật toán tìm kiếm mù (Blind/ Uninformed Search):

1.1. Thuật toán BFS (Breadth First Search)

Thuật toán tìm kiếm theo chiều rộng BFS là thuật toán sử dụng cấu trúc dữ liệu hàng đợi (FIFO), từ một node cho trước duyệt các node kề với node vừa cho có thể hướng tới. BFS thường được sử dụng cho hai mục đích: tìm kiếm đường đi từ node gốc tới đích hoặc tìm đường đi từ node gốc đến tất cả node khác.

Pseudocode:

```
create frontier to store node (frontier is queue)
add node start to frontier
create came_from to store visited node
came_from(start) = none
until frontier is empty:
    current = first node of frontier
    if current = goal
        End
    remove current from frontier
    for each node is neighbors of current
        if node not in came_from:
            add node to frontier
            came_from(node) = current
```

Đánh giá:

- Độ phức tạp về mặt thời gian và không gian: $O(b^d)$ với d là chiều dài đường đi tới đích.
- Mang tính vét cạn, tốn nhiều không gian lưu trữ.
- Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm, tìm nhiều lời giải.
- Chắc chắn tìm ra lời giải nếu có.
- Tối ưu khi chi phí di chuyển như nhau. Chi phí: tính từ node trước đến node hiện tại.
- Phát triển xu hướng tham lam: mỗi bước chọn đường đi ngắn từ node đang xét tới node khác.

1.2. Thuật toán DFS (Depth First Search)

Thuật toán tìm kiếm theo chiều sâu DFS là thuật toán gần tương tự như BFS nhưng sử dụng cấu trúc dữ liệu ngăn xếp (LIFO), thay vì mở các node có thể ở cùng cấp, DFS ưu tiên mở các node theo hướng xa nhất có thể theo mỗi nhánh.

```
create frontier to store node (frontier is queue)
add node start to frontier
create came_from to store visited node
came_from(start) = none
until frontier is empty:
    current = last node of frontier
    if current = goal
        End
    remove current from frontier
    for each node is neighbors of current
        if node not in came_from:
            add node to frontier
            came_from(node) = current
```

Đánh giá:

- Độ phức tạp về mặt không gian: $O(bm)$, về mặt thời gian $O\left(b^m\right)$ với m là độ sâu tối đa của cây tìm kiếm.
- Thuật toán không tối ưu khi m quá lớn so với b.
- Sử dụng bộ nhớ tối ưu hơn so với BFS.
- Tìm được lời giải nếu có (nếu không gian tìm kiếm hữu hạn).
- Tối ưu: khi đích thuộc tập những node được thêm sau cùng (nằm ở đỉnh ngăn xếp).

1.3 Thuật toán UCS (Uniform Cost Search)

Thuật toán tìm kiếm chi phí đồng nhất là thuật toán chỉ quan tâm đến chi phí đường đi, mỗi bước chọn mở node có chi phí đường đi thấp nhất. Là thuật toán tối ưu về mặt chi phí (tìm đường đi ngắn nhất).

```
create frontier to store node (priority queue)
add node start and its cost in frontier          #(0, start)
create came_from to store node visted
came_from(start) = none
create cost to store node's cost
cost(start) = 0
until frontier is empty
   current = least-cost-node in frontier
   if current = goal:
      End
   for each node is neighbor of current
      new_cost = path_cost(current, node)
      if node is not in cost or new_cost < cost(node):
         cost(node) = new_cost
         add node and cost to frontier     #(cost(node), node)
         came_from(node) = current
```

Đánh giá:

- Độ phức tạp về mặt không – thời gian: $O\left(b^{1+\left[C^*/\varepsilon\right]}\right)$ với C^* là chi phí lời giải tối ưu, ε là chi phí di chuyển thấp nhất.
- Chắc chắn tìm được lời giải (khi chi phí đường đi lớn hơn 0).
- Khi chi phí ở mỗi bước là như nhau thì thuật toán trở thành BFS.
- Lời giải tối ưu.

2. Thuật toán tìm kiếm có thông tin (Informed Search):

Trước khi đi vào đánh giá thuật toán, ta sẽ tóm lược hai hàm heuristic (hàm đánh giá) cho thuật toán. Có hai hàm heuristic được áp dụng cho thuật toán tìm kiếm GBFS và A*, bao gồm:

- **Hàm khoảng cách Euler**: Hàm tính đường nối giữa hai điểm trong không gian, cụ thể tại đây ta dùng hàm này để tính khoảng cách nối từ một node đến đích theo công thức:
  $\sqrt{\left(x_1-x_2\right)^2+\left(y_1-y_2\right)^2}$

- **Hàm khoảng cách Manhattan**: còn gọi là khoảng cách L1 hay khoảng cách thành phố, bằng tổng trọng số chiều dài của hình chiếu của đường thẳng nối hai điểm trong không gian, ta cũng dùng hàm này để tính khoảng cách giữa một node và đích theo công thức:
  $\left|x_1-x_2\right|+\left|y_1-y_2\right|$

Trong đó, $\left(x_1,\mathrm{y}_1\right),\left(x_2,y_2\right)$ lần lượt là tọa độ của điểm node đang xét và điểm đích. Về cơ bản, khoảng cách Euler sẽ cho kết quả tốt (nhỏ) hơn khoảng cách Mahattan nếu $x_1 \neq x_2$ và $y_1 \neq y_2$.

2.1 Thuật toán GBFS (Greedy Best First Search)

Thuật toán tìm kiếm tham lam GBFS là thuật toán cố gắng khám phá tất cả các node gần với đích nhất có thể, được bổ sung tri thức bằng cách sử dụng hàm heuristic *h(n)* để đánh giá mỗi node. Với mỗi node được mở, thuật toán tìm kiếm các node lân cận gần nhất và chọn node lân cận có khoảng cách gần đích hơn các node còn lại, quá trình lặp lại đến khi tìm được đích.

Pseudocode:

```
create frontier to store node
frontier.append(start)
create came_from to store visited node
came_from(start) = none
until frontier is empty:
   current = 1st node of frontier
   for each node in frontier:
      if h(node) < h(current):   #h(n) is heuristic function
         current = node
   if current is goal:
      End
   frontier.remove(current)
   for each neighbor of current:
      if neighbor is not in came_from:
```

```
        frontier.append(neighbor)
        came_from(neighbor) = current
```

Đánh giá:

- Độ phức tạp về thời gian và không gian của GBFS là $O\left(b^m\right)$ với *m* là độ sâu tối đa của không gian tìm kiếm
- Tìm ra lời giải nếu có
- Thuật toán có thể không tối ưu vì mỗi bước chỉ quan tâm đến node nào gần đích nhất
- Tuy nhiên tùy bài toán cụ thể và hàm đánh giá tốt có thể cho một lời giải tối ưu.

2.2 Thuật toán A*

Thuật toán tìm kiếm A* cũng là một dạng tìm kiếm theo lựa chọn tốt nhất phổ biến nhất, cũng sử dụng một hàm đánh giá heuristic *h(n)* để đánh giá mỗi node khi duyệt. A* là thuật toán "pha trộn" giữa UCS và GBFS, khi điểm khác biệt là hàm heuristic của thuật toán sử dụng cả chi phí đường đi đã đi qua và khoảng cách ước tính từ node đang xét đến đích để đánh giá mỗi node. Điều này làm thuật toán A* "đầy đủ" và "tối ưu" hơn so với UCS và GBFS.

Pseudocode:

```
create frontier to store node
frontier.append(start)
create came_from to store visited node
came_from(start) = none
create cost to store path cost of each node
cost(start) = 0
until frontier is empty:
   current = 1st node of frontier
   for each node in frontier:
      if cost(node) + h(node) < cost(current) +h(current):   #h(n) is heuristic function
         current = node
   if current is goal:
      End
   frontier.remove(current)
   for each neighbor of current:
       new_cost  = path_cost(current, neighbor)
       if neighbor is not in cost or new_cost < cost(neighbor)
          cost(neighbor) = new_cost
          frontier.append(neighbor)
          came_from(neighbor) = current
```

Đánh giá:

- Độ phức tạp về mặt không – thời gian phụ thuộc vào chi phí di chuyển và không gian tìm kiếm của thuật toán.
- Chắc chắn tìm được lời giải nếu chi phí đường đi lớn hơn 0.
- Tối ưu với hàm đánh giá hợp lý (không ước lượng quá cao chi phí đến đích) và nhất quán (ước tính từ node cha đến đích không quá chi phí đường đi từ node cha đến node con + ước tính từ node con đến đích).
- Có hiệu suất tốt hơn các thuật toán tìm kiếm có thông tin trên.

3. Bảng đánh giá thuật toán với từng trường hợp:

Thuật toán tối ưu đường đi: X

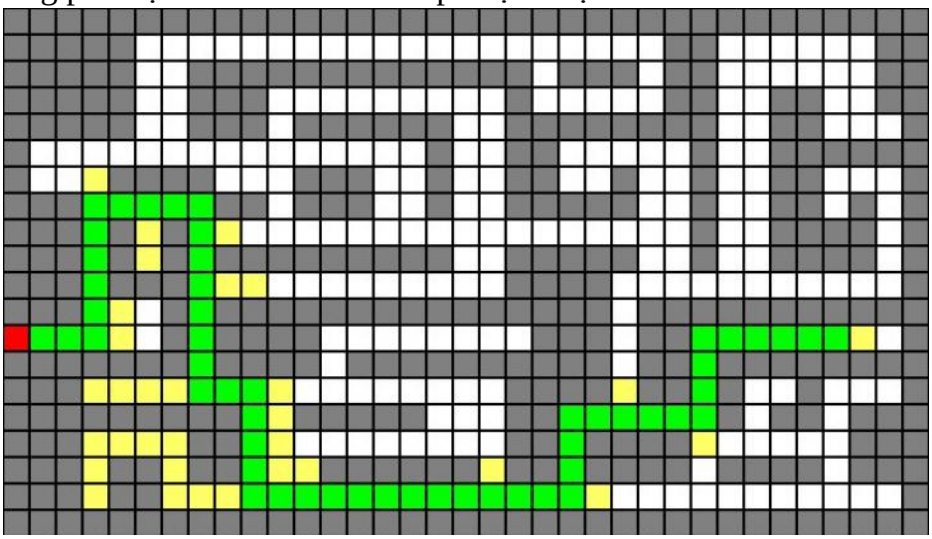
Thuật toán chưa tối ưu đường đi: Để trống

Mê cung	**Thuật toán**						
	BFS	**DFS**	**UCS**	**GBFS** (Euler)	**GBFS** (Manhattan)	**A*** (Euler)	**A*** (Manhattan)
1	**X**		**X**			**X**	**X**
2	**X**		**X**			**X**	**X**
3	**X**	**X**	**X**	**X**	**X**	**X**	**X**
4	**X**		**X**	**X**	**X**	**X**	**X**
5	**X**		**X**			**X**	**X**

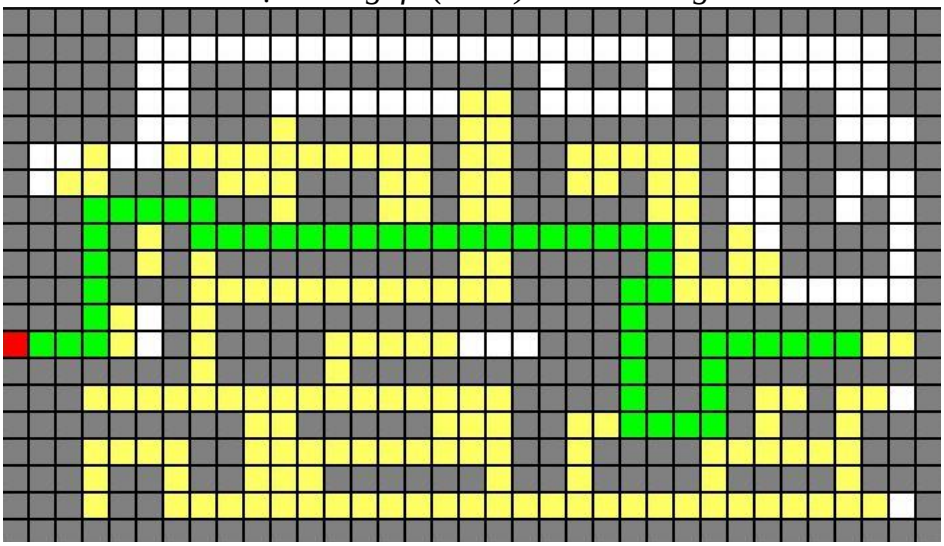
Vì chi phí đường đi luôn lớn hơn 0 và chi phí mỗi bước là như nhau nên thuật toán BFS, UCS và A* luôn tìm được đường đi ngắn nhất trong 5 trường hợp.

Xét **mê cung 1:**

Vì thuật toán GBFS chỉ cố gắng tìm đường đi càng gần đích càng tốt nên tại node giao đã bỏ qua đường đi có chi phí tối ưu hơn. Tuy nhiên tốc độ di chuyển đến đích của GBFS rất nhanh và lời giải không phải tệ nhất nên có thể chấp nhận được.

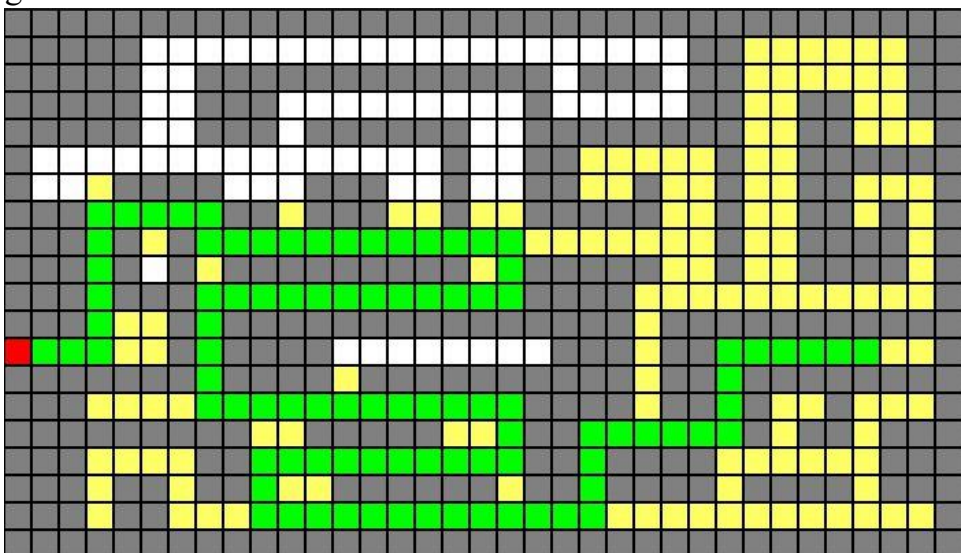


Thuật toán gbfs (euler) cho mê cung 1



Thuật toán A (euler) với chi phí thấp hơn*

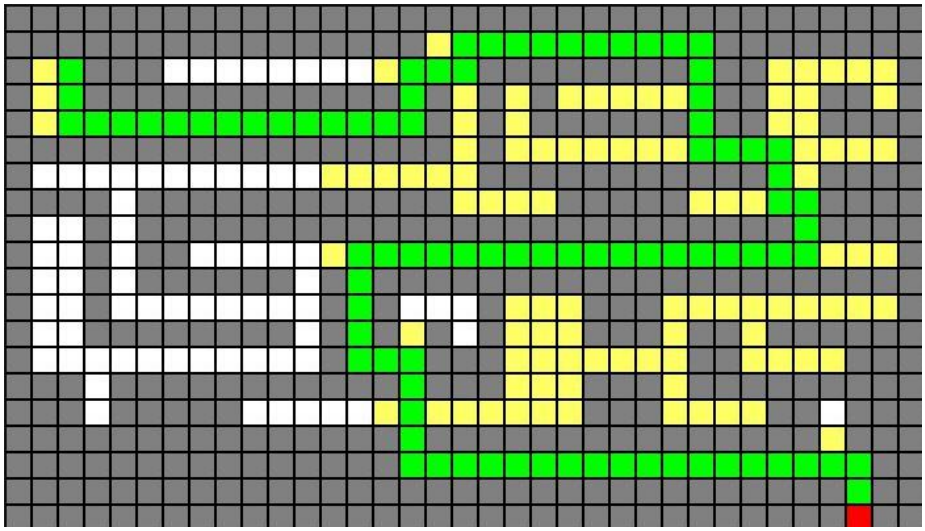
Thuật toán DFS lại không tối ưu trong trường hợp này vì chọn mở node và hướng đi quá xa so với đường đi tối ưu.



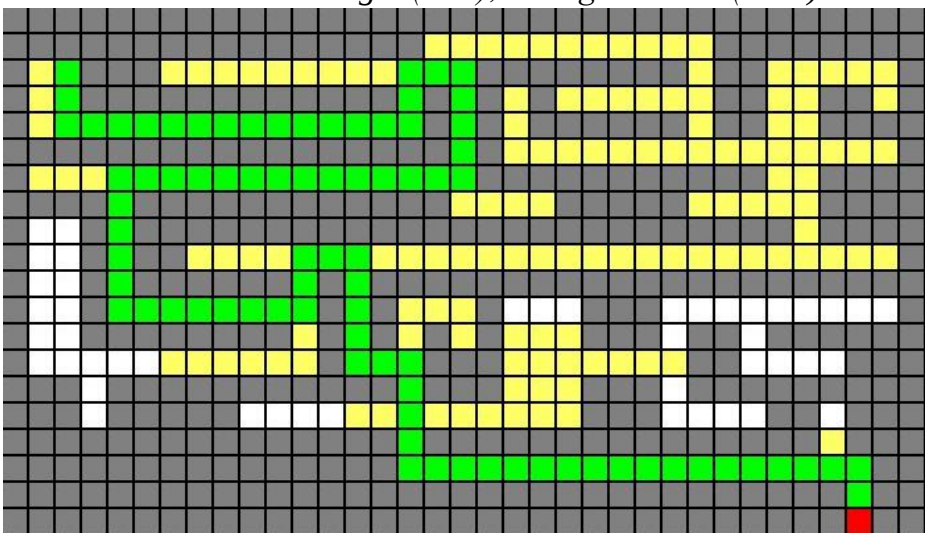
Thuật toán DFS cho mê cung 1

Xét **mê cung 2:**

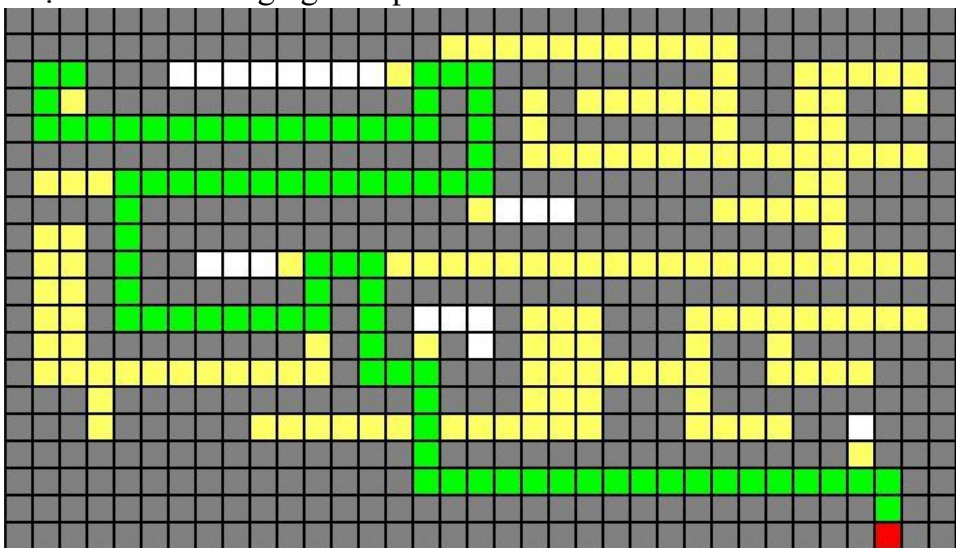
Tương tự như mê cung 1, GBFS chọn mở các nút càng gần đích càng tốt nên bỏ qua đường đi tối ưu.



GBFS cho mê cung 2 (trên), đường đi tối ưu (dưới)

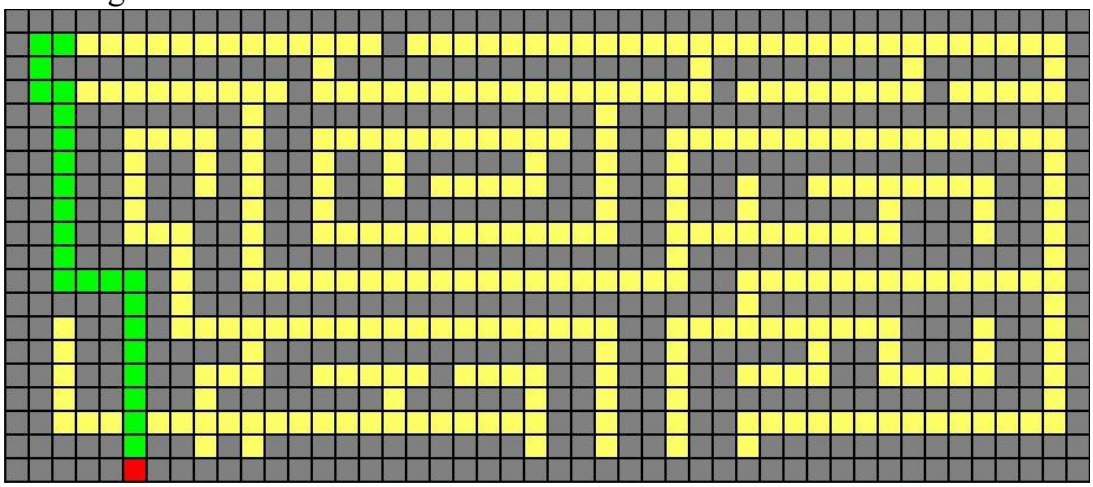


Thuật toán DFS chọn được hướng đi đúng nhưng vẫn không tối ưu được chi phí đường đi vì thứ tự node được đưa vào trong ngăn xếp.



Xét **mê cung 3:**

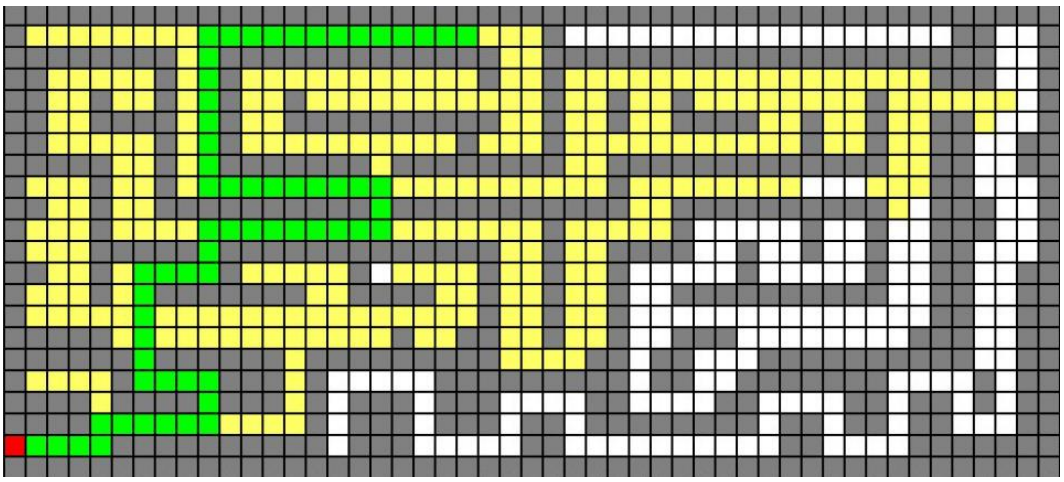
Ở mê cung 3, tất cả thuật toán đều đưa ra lời giải tối ưu (vì đường đi này là duy nhất), sự khác biệt giữa các thuật toán nằm ở tốc độ tìm kiếm. Nhanh và tối ưu nhất là thuật toán GBFS bởi chiến thuật tham lam, gần như tương tự GBFS về mặt thời gian là A*. BFS và UCS có tốc độ trung bình chấp nhận được. DFS chọn hướng duyệt chưa tốt và phải duyệt toàn bộ mê cung để tìm ra đường đi cuối.



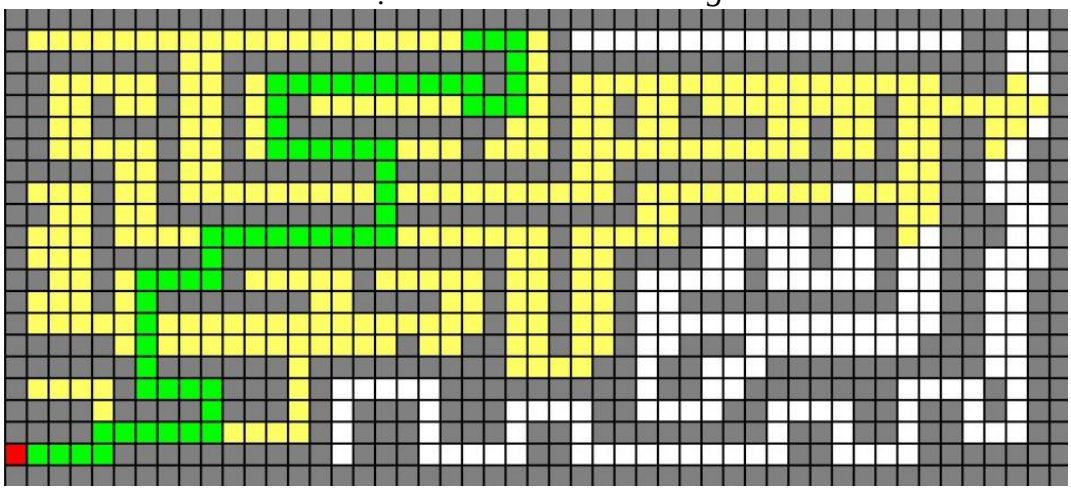
Thuật toán DFS cho mê cung 3

Xét **mê cung 4:**

Cả ba thuật toán BFS, GBFS và A* cho lời giải đường đi giống nhau, nhưng GBFS và A* vẫn là những thuật toán có tốc độ giải tốt hơn. UCS cho một lời giải khác BFS vì cách tính toán chi phí đường đi khác nhau nhưng vẫn cho lời giải với chi phí đường đi tối ưu. DFS không phù hợp với trường hợp này (giống mê cung 1) vì cho lời giải với chi phí cao hơn nhiều so với các thuật toán còn lại.



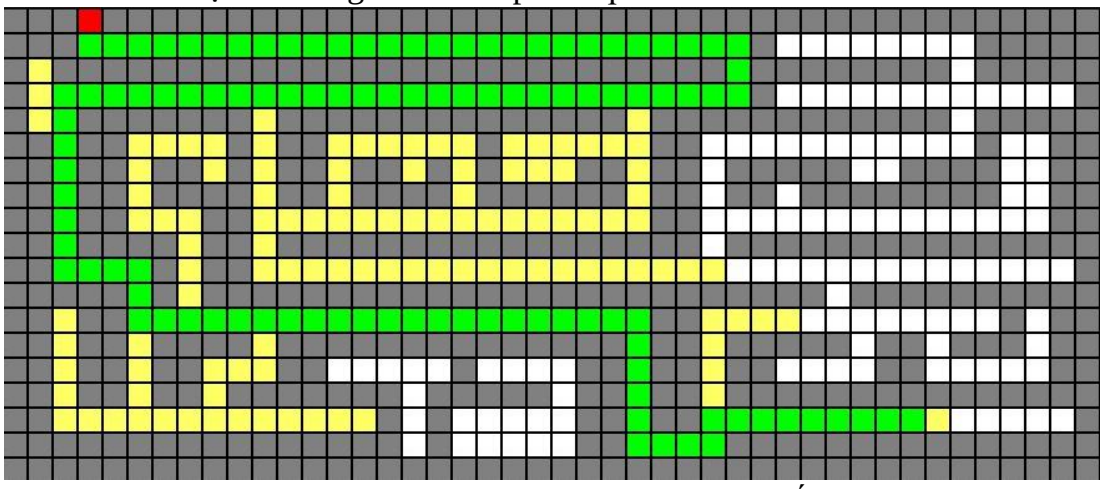
Thuật toán BFS cho mê cung 4



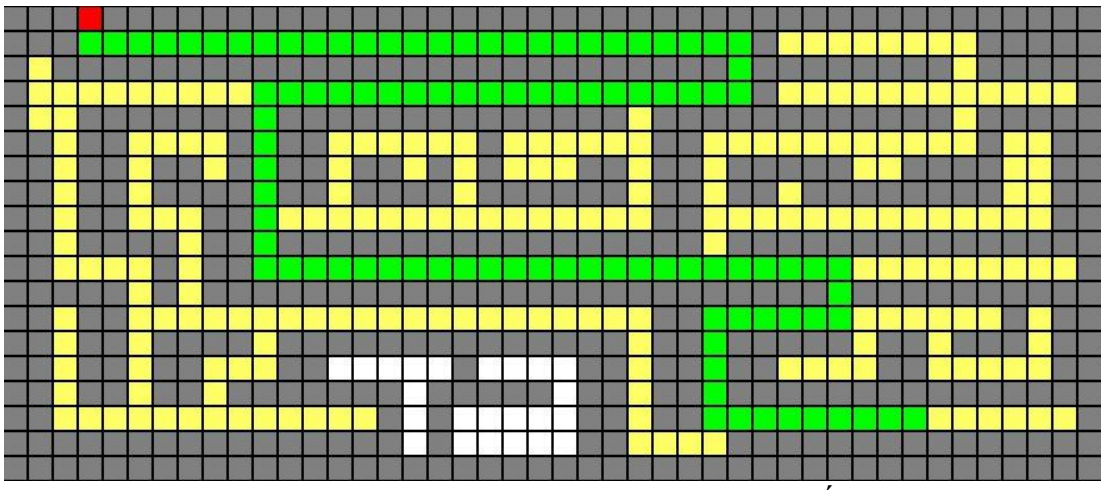
Thuật toán UCS cho mê cung 4 (cho lời giải khác)

Xét **mê cung 5:**

Ba thuật toán BFS, UCS và A* cho lời giải như nhau, A* có tốc độ giải tốt hơn hai thuật toán còn lại. DFS chọn hướng đi giống với GBFS nhưng cả hai đều bỏ qua việc duyệt tiếp từ những node đã thăm có cơ hội cho lời giải với chi phí thấp hơn.



Thuật toán GBFS cho mê cung 5(lời giải giống DFS)



Thuật toán A cho mê cung 5 cho lời giải tối ưu*

Nhìn chung, BFS và UCS hầu hết cho kết quả tương tự nhau. Đối với hàm heuristic cho tìm kiếm đường đi có thông tin, hai hàm đánh giá không có sự khác biệt lớn, tuy nhiên hàm khoảng cách Euler sẽ có tốc độ giải nhanh hơn một chút so với Manhattan với một số trường hợp cụ thể (khả năng mở node nhiều hơn). A* là thuật toán có thể tối ưu cả về chi phí đường đi và tốc độ duyệt trong tất cả 5 trường hợp.

IV. Cấu trúc bài nộp

1. Input

3 thư mục **level_1, level_2** (trống), **advance** (trống). 5 files **input<num>.txt** tương ứng với 5 mê cung ở thư mục **level_1**.

2. Output

Sẽ gồm các thư mục tương ứng với từng level và từng input, mỗi input sẽ gồm các file **.txt** chứa chi phí đường đi của thuật toán và file **.jpg** chứa lời giải thuật toán.

Quá trình xuất lời giải có thể tham khảo ở video sau: Search_Algorithms

3. Source

Thư mục source chứa duy nhất một file **main.py** thực thi toàn bộ thuật toán.

4. Run.sh

Định dạng với các tham số dòng lệnh theo format:

python ./source/main.py <level_num> <input_num> <algorithm>

trong đó **<algorithm>** là các thuật toán áp dụng cho mê cung.

5. Report

File báo cáo quá trình làm đồ án của nhóm **(.pdf)V. Bảng tự đánh giá hoàn thành**

MSSV	Họ và tên	Mức độ hoàn thành
20120263	Phạm Võ Hải Đăng	100%
20120279	Trương Cao Hoàng Gia	95% (Hàm heristic chưa có nhiều khác biệt)
20120289	Võ Minh Hiếu	100%

VI. Tài liệu tham khảo

1. Lê Hoài Bắc, Tô Hoài Việt, 2014, Chương 1 & 2, *Giáo trình Cơ sơ trí tuệ nhân tạo*
2. Wikipedia, sửa đổi lần cuối 09/09/2021, *Tìm kiếm theo chiều rộng*
3. Wikipedia, sửa đổi lần cuối 13/02/2022, *Tìm kiếm theo chiều sâu*
4. Wikipedia, sửa đổi lần cuối 23/08/2020, *Tìm kiếm chi phí đều*
5. Wikipedia, sửa đổi lần cuối 06/09/2021, *Giải thuật tìm kiếm A**
6. pp_pankaj, 23/08/2022, *Difference between Informed and Uninformed Search in AI*