ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

20127110 – PHAN HỮU ĐOÀN ANH

20127512 – TRẦN QUỐC HƯNG

20127643 - TRƯƠNG GIA TIẾN

PROJECT 1 – SEARCH - I

Finding path by searching algorithm

| Giáo viên hướng dẫn |

TS. Nguyễn Hải Minh

TS. Nguyễn Ngọc Thảo

Thầy Nguyễn Thái Vũ

Môn: Cơ sở trí tuệ nhân tạo

I. Table of content

II. Introduction

Searching is one of the widespread problems and has been constantly developed since the beginning of the development of computers and has become one of the popular tools to help people in life. A search problem consists of five components: state space, initial state, sub-state function, target state set, and path cost function.

Common search problems include two groups: game problems (8-puzzle, eight puzzles) and practical problems (Find the way, the shopkeeper, the robot finds the way)

In this Lab, we will perform the problem of finding the way to the destination location in the 2-dimensional matrix of the pathfinder robot with the ability to move up and down left and right with the obstacles being the given polygons.

- The algorithms are:

- Breadth-first Search - BFS
- Uniform-cost Search - UCS
- Iterative Deepening Search - IDS
- Greedy Best-first Search
- A-star

III. Describe

- We used Turtle and NumPy in this project.
- Create Turtle.
- Read the data from the input file to draw the matrix, and the polygon.
- By calling the algorithm function, the program will return the matrix and the polygon to the screen.
- The turtle start finding path based on the algorithms we called.

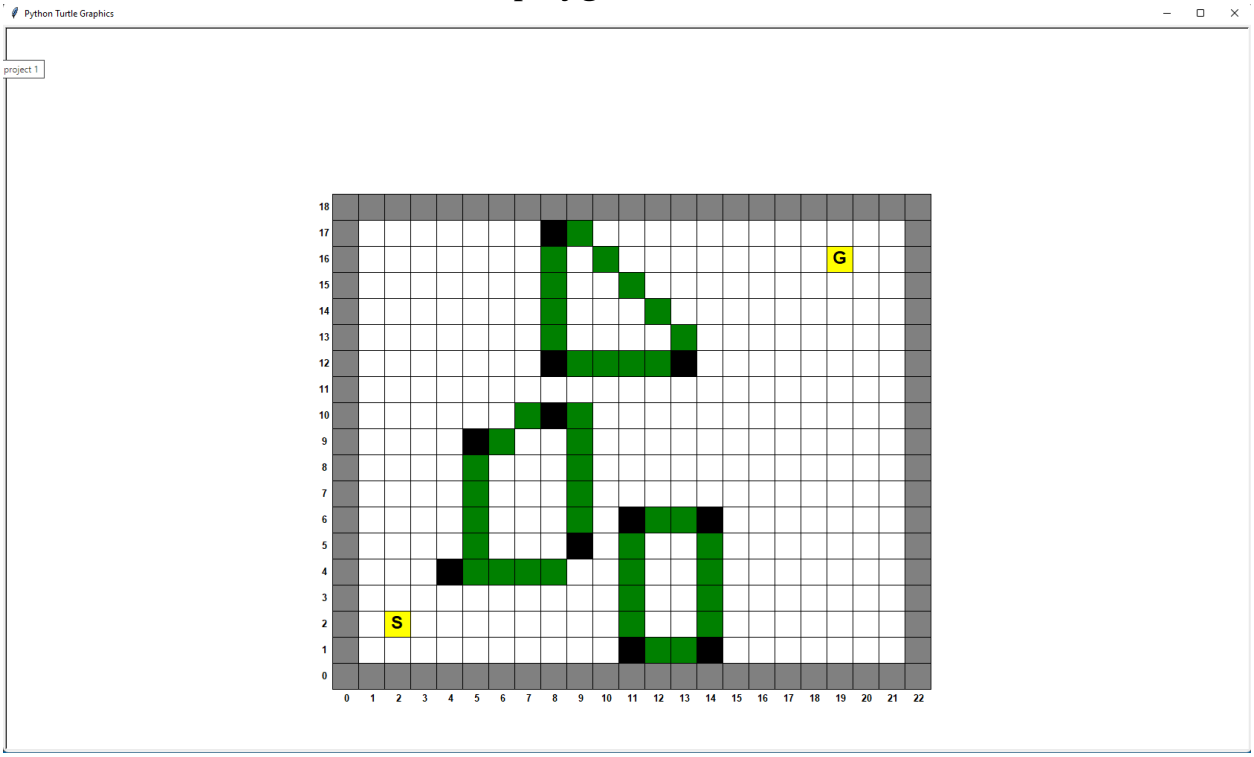
IV. Algorithms

Common set-up:

- Read the information on matrix and polygon from the input file.

```
22 18
2 2 19 16
3
4 4 5 9 8 10 9 5
8 12 8 17 13 12
11 1 11 6 14 6 14 1
```

- Draw a 2-dimensional matrix and polygon with the turtle.

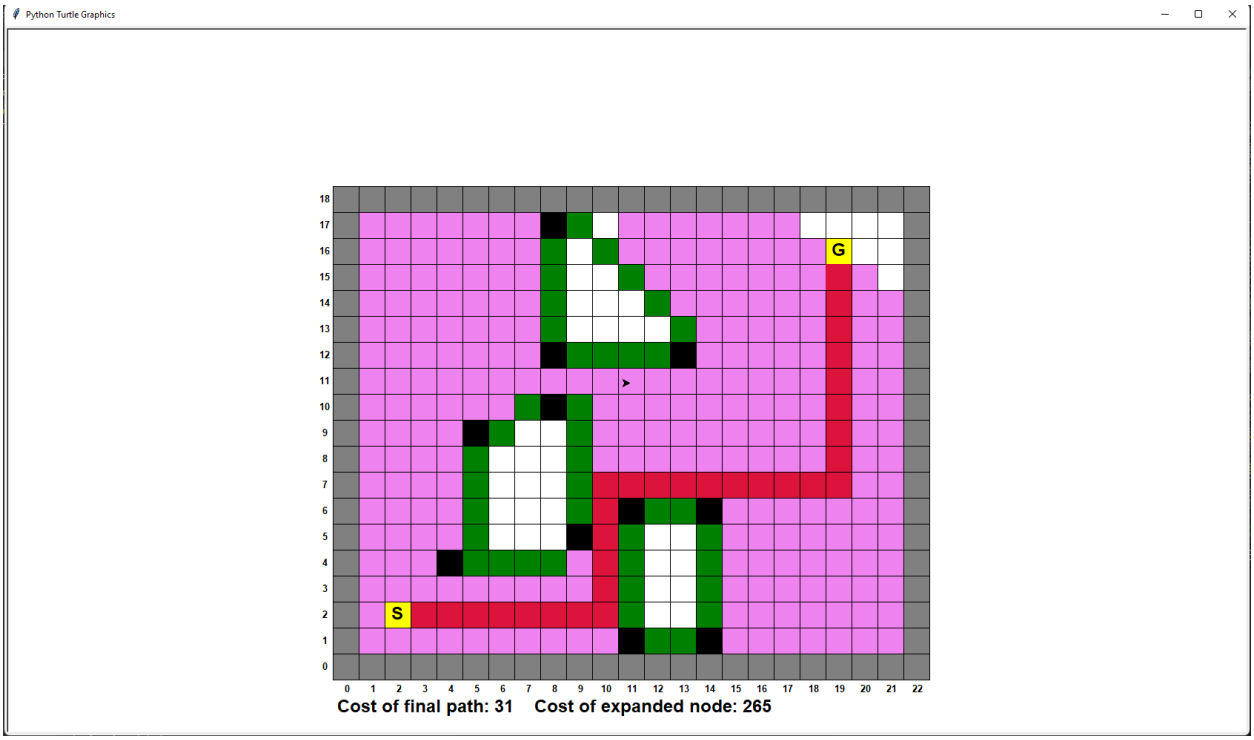


  - S is Start and G is Goal.
  - A pixel in the polygon that is fill in black is origin coordinates.

a) Breadth-first Search:

- Input: Common set-up
- Initialize queue to save tuple of the coordinates, the cost, and the finding-path.
- Initialize the 2-matrix direction to save the next move of the turtle.
- Initialize the 2-matrix visited to save the node that passed.
- In the while loop:
    - Stop if the queue is empty.
    - Pop the coordinates.
    - If we find the goal, draw the finding path, and return the cost of the absolute path and cost of the expanded node.
    - If the coordinate can go, then update the 2-matrix and draw it by the turtle.
    - Traverse direction in up, down, left, and right. If the following coordinates are not a polygon or out of the matrix, then update visited and save it to queue (add to frontier).

Output:



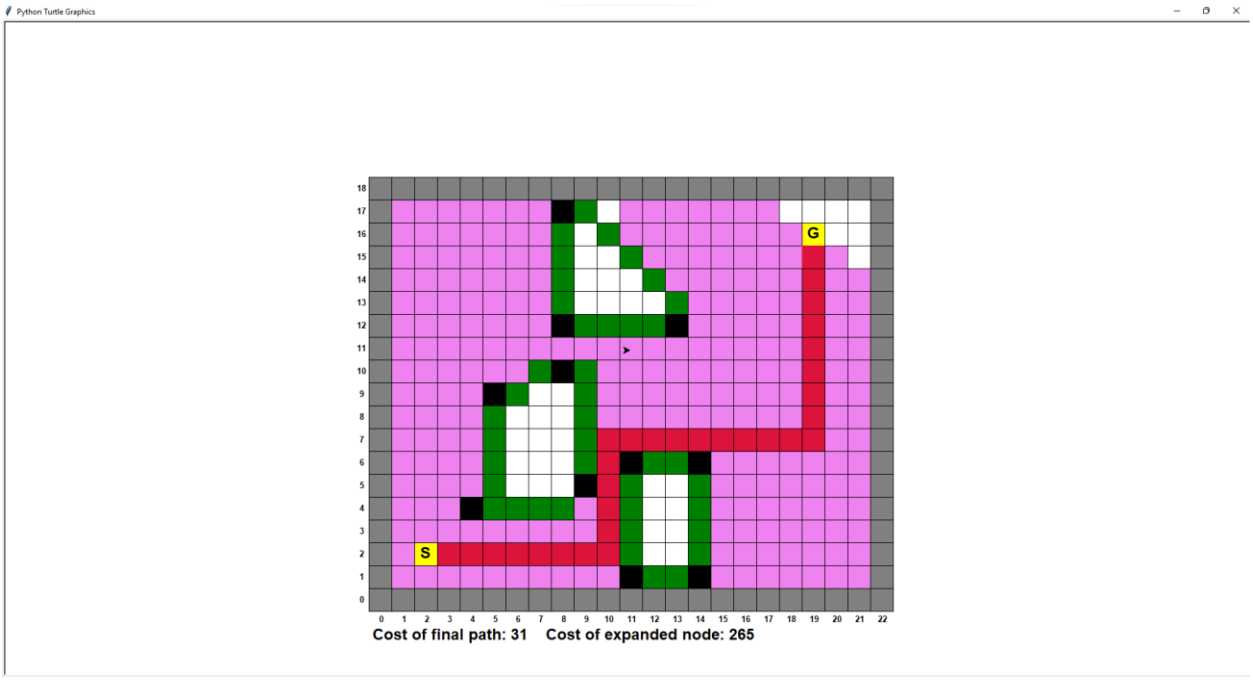
Conclusion:

- High run time.
- Easy to understand.
- The cost depends on the priority of moving direction.

b) Uniform-cost Search – UCS:

- Input: Common set-up
- initialize priority queue as a frontier to save the coordinates, the finding path, and cost.
- Initialize the 2-matrix way to save the movement of the turtle.
- initialize the 2-matrix visited to save the node that passed.
- In the while loop:
    - Stop if the queue is empty.
    - pop the first coordinate from the frontier and flag the coordinate passed.
    - Traverse the moment matrix and set it on a new coordinate.
    - If the new coordinates are the goal, traverse the direction and set the turtle at the star. Initialize the cost, and the cost visited. For every move, the cost will be plus by one and update the coordinate till the turtle reach the goal. Print the cost and the cost visited on the screen.
    - If the new coordinates are not the goal, flag the coordinated passed and update the frontier and pathfinding based on the movement matrix.
- Calculate all the visited coordinate costs.

Output:



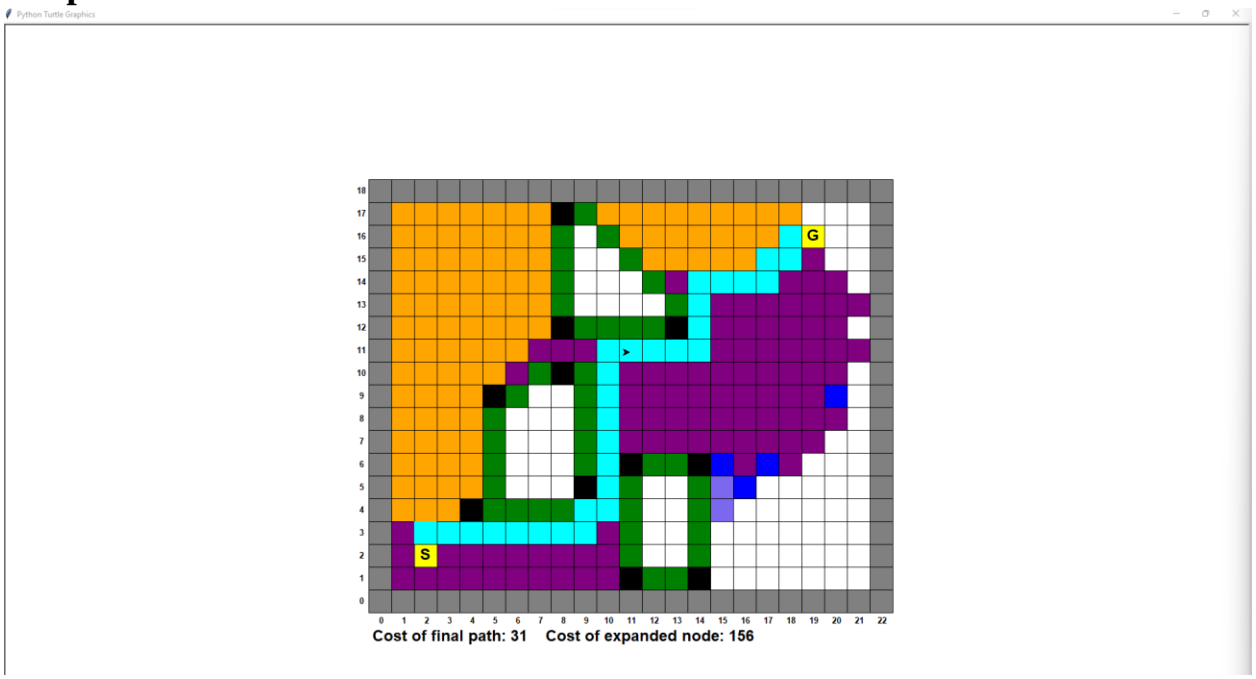
Conclusion:

Because there is no weight when traversing the terrain, the UCS algorithm only needs to find the shortest path to the destination corresponding to the set priority of the agent's direction. So go as far to the right and up as we can.

c) Iterative Deepening – Search (IDS):

- Input: Common set-up
- The IDS algorithm helps limit the depth of consideration so that it can traverse all cases from which it is possible to enter the branch to the result. need to find better without wasting money having to drill down into the branch that is not necessarily the result
- Using DLS to create IDS algorithms because DLS has the same core as DFS.
- DLS might not find the goal if the limit < truth depth, so creating IDS, we set the limit increase until we find the goal, but if the limit is over the matrix size, stop.
- Initialize frontier to save coordinates and Initialize solution Global to save finding-path.
- In the while loop:
    - If the frontier is empty, stop looping.
    - If the coordinate at the goal, then return true and jump to IDS function
    - If the limit equals zero and does not find the goal, continue the loop.
    - If the next move of the coordinate is valid, then add it into the frontier.
- After looping in DLS and not finding the goal, increase the limit and continue DLS with a new limit until found the goal but cannot over the matrix size.

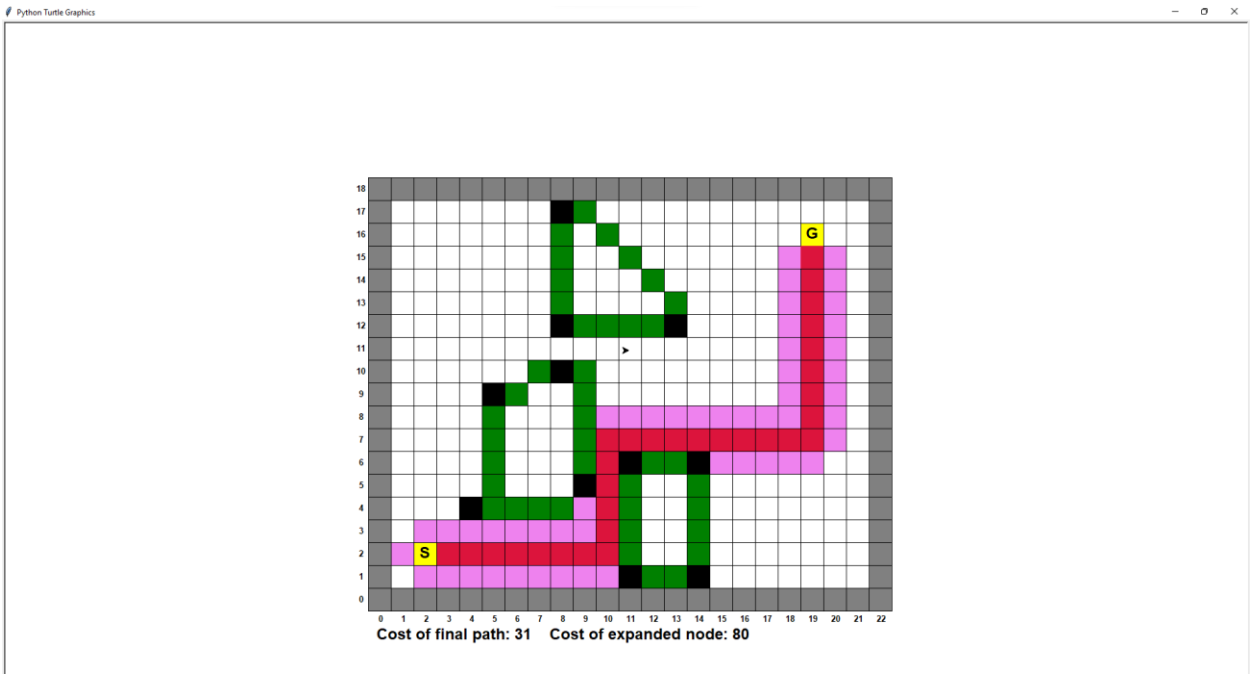
Output:



d) Greedy Best-first Search (G-BFS):

- Input: Common set-up
- Idea: Because the heuristic function returns an estimate of the distance from the current position to the destination, it only needs to go to the right or up to reach the optimal position, so there is no need to consider and calculate the smallest heuristic to expand to there.
- initialize priority queue as a frontier to save the coordinates, the finding path, and cost.
- Initialize the 2-matrix way to save the movement of the turtle.
- initialize the 2-matrix visited to save the node that passed.
- In the while loop:
    - pop the queue to take the current coordinate and create the next move of the current coordinates.
    - if the next move is the goal, take the path to the goal and start with the start again. For every movement, the cost will be plus by one till we reach the goal and increase the visited cost for the last time.
    - If the next move is neither the goal nor the polygon, then update the coordinate passed and the new move into the frontier.
- Calculate all the visited coordinate costs.

Output:



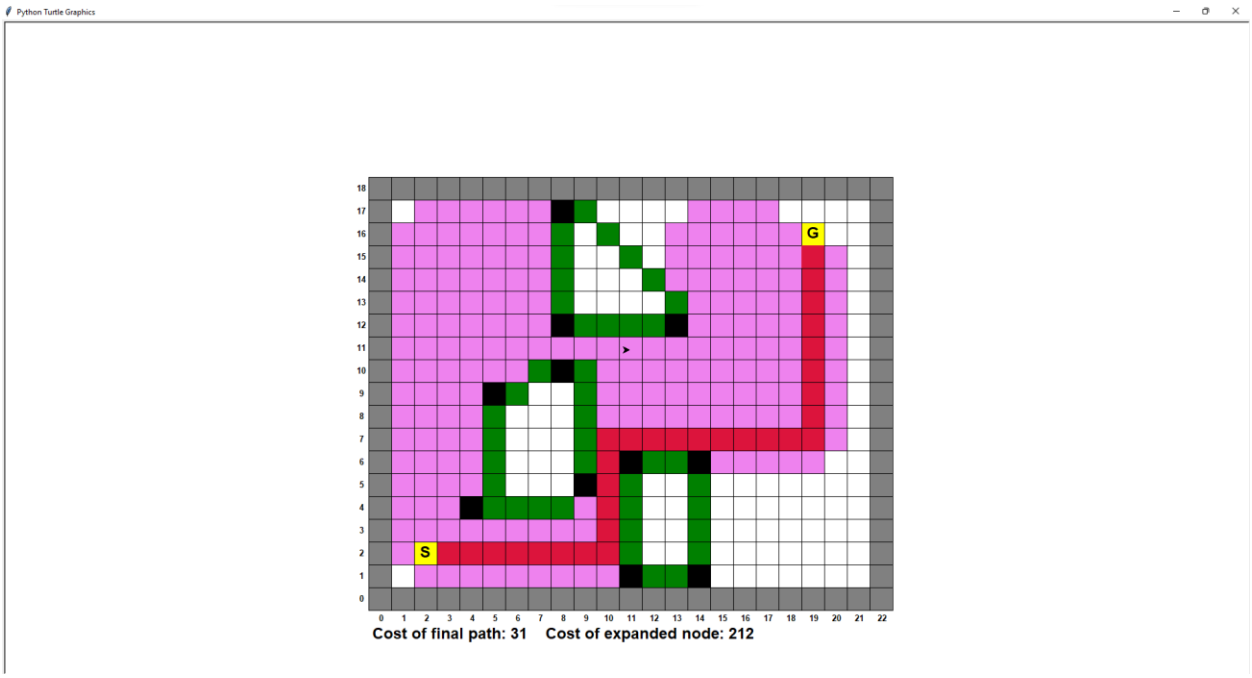
Conclusion:

Because the algorithm works on Heuristic units, the running time will depend on the Heuristic calculation function, the closer the calculation function is to h*(n), the faster the algorithm will run.

e) A-star (A*):

Idea: As a combination of BFS and UCS, the only difference is the cost to the destination is the result of the heuristic function + path cost.

Output:



Conclusion:

This combines BFS and UCS to calculate the f(n) value in the frontier for the shortest path. It must also estimate that the next step would be low-cost, so the overlay map of A* will be general because there is no terrain on the way.

V. References:

https://www.youtube.com/watch?v=dv1m3L6QXWs

https://www.redblobgames.com/pathfinding/a-star/implementation.html
https://docs.python.org/3/library/turtle.html#turtle.up
http://aimacode.github.io/aima-javascript/3-Solving-Problems-By-Searching/
https://www.w3schools.com/python

https://www.youtube.com/watch?v=JtiK0DOeI4A&
https://www.youtube.com/watch?v=4wgRlNqIKqM
Giáo trình cơ sở Trí tuệ Nhân tạo Khoa Công nghệ Thông tin

Tài liệu bài giảng môn Cơ sở Trí tuệ Nhân tạo