VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

---------------o0o---------------

REPORT: SEARCH

Course: **INTRODUCTION TO ARTIFICIAL INTELLIGENCE**

Subject: **SEARCH**

Lecture: **BUI TIEN LEN**

Teaching assistant: **NGUYEN THAI VU**

Student: **LE NGOC TUONG 20127383**

Class: **20CLC10**

HO CHI MINH CITY –2022

THANK YOU

I would like to express our sincere thanks to teacher Bui Tien Len and Nguyen Thai Vu for helping us throughout the learning process to complete this project, teachers for creating conditions for me to have a good and useful subject to expand our knowledge, helping us to improve in the process study.

However, in the process of researching the topic, specialized knowledge is still limited, so I still has many shortcomings in understanding, evaluating and presenting the topic, looking forward to receiving the attention and suggestions of the teachers let my topic be completed.

I wish you all the best health and success in your career. Once again, I would like to thank the teachers for their attention and help.

I. STUDENT'S INFORMATION:

Name: Le Ngoc Tuong.

Student code: 20127383.

Class: 20CLC10.

II. WORKING ENVIRONMENT:

Python 3.10.0.

Visual Studio Code & Github: https://github.com/lengoctuong/Search_Introduce-to-AI

III. INTRODUCE TO PROGRAM:

1. Run program:

Input: algorithms (choose 1 of 5 algorithms) and modes (0 or 1). Mode 0 (run fast) only find path and mode 1 (run slowly) step by step to find path.

Program draw and fill color maze, source, goal and obstacles.

Program find a path from source to goal according to mode which user choose.

Output: graphical representation of polygons and path and cost (from source to goal).

User press 'Enter' to end program.

2. Functions and classes:

a) *Global constants and variables:*

b) *Sub functions:*

`def read_input_file(fileName): Read 'fileName'.`

`def rectangle(width, height): Draw Rectangle with (width, height).`

`def fill_block(coordinate, color): Fill 'color' for a Block at 'coordinate'.`

`def mark_block(coordinate, symbol, color): Mark a 'symbol' for a Block at 'coordinate' with color is 'color'.`

`def drawFillColorSourceGoal(sourceState, sourceLabel, goalState, goalLable, blockColor, labelColor): Draw 'sourceLabel', 'goalLabel' at source coordinate ('sourceState'), goal coordinate ('goalState') with block color is 'blockColor' and label color is 'labelColor'.`

def add_obstacles(obstacle1, obstacle2, polygons, color): Add new Obstacles between ‘obstacle1’, ‘obstacle2’ and fill ‘color’ it.

c) *Classes:*

class Problem:

- Problem of task.
- Data: width (width of maze), height (height of maze), sourceState (source coordinate), model (coordinate of obstacles), goalState (goal coordinate), pathCost (cost to forward from one coordinate to adjacent one, always equal 1).
- Function:
    - def actions(self, state): Calculate actions can have from one ‘state’.
    - def result(self, state, action): Calculate result of new state by ‘action’.

class Node:

- One coordinate of maze.
- Data: state (coordinate of Node), parent (parent node, type is ‘Node’), action (action to forward from parent node to this node), pathCost (cost for move from source to this node).

d) *Main functions:*

def child_node(problem, parent, action): Move to new node from ‘node’ with 'action'.

def nodeHaveInFrontier(node, frontier): Check whether ‘node’ have in ‘frontier’.

def HeuristicFunction(node, goalState): Calculate manhattan distance (h(n)) between ‘node’ and ‘goalState’.

def EvaluationFunction(node, goalState): Calculate evaluation (f(n) = g(n) + h(n)) between ‘node’ and ‘goalState’.

def popMinEvaluation(search, goalState, frontier): Pop node in ‘frontier’ with min evaluation (‘search’ is name of algorithms), (for BFS (pop head), UCS (pop min path cost), GBFS (pop min heuristic) and ASS (pop min evaluation)).

def nodeHaveInFrontierWithHigherEvaluation(search, node, goalState, frontier): Check whether ‘node’ have in ‘frontier’ and node in frontier have higher evaluation this node (‘search’ is name of algorithms), (for UCS (check for higher path cost) and ASS (check for higher evaluation)).

def replaceNodeWithHigherEvaluation(search, node, goalState, frontier): Replace ‘node’ have higher evaluation and same state with other node in ‘frontier’ (‘search’ is name of algorithms), (for UCS (replace higher path cost) and ASS (replace higher evaluation).

def BFS(problem, mode): Breadth-first search solve ‘problem’, ‘mode’ is 0 or 1.

def UCS(problem, mode): Uniform-cost search solve ‘problem’, ‘mode’ is 0 or 1.

`def IDS(problem, mode): Iterative depening depth-first search solve ‘problem’, ‘mode’ is 0 or 1.`

`def GBFS(problem, mode): Greedy best-first search solve ‘problem’, ‘mode’ is 0 or 1.`

`def ASS(problem, mode): A* search solve ‘problem’, ‘mode’ is 0 or 1.`

IV. ALGORITHMS:

Note:

- *Priority move order: up -> right -> down -> left.*
- *Frontier is queue.*
- *Example for each algorithms is in lab 1; with green: explored, purple: frontier, ‘+’: path, mode: 1).*

1. Breadth-first search:

a) *Idea: root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded*

Check whether initial node is goal node, then return.

Else, add this node to frontier.

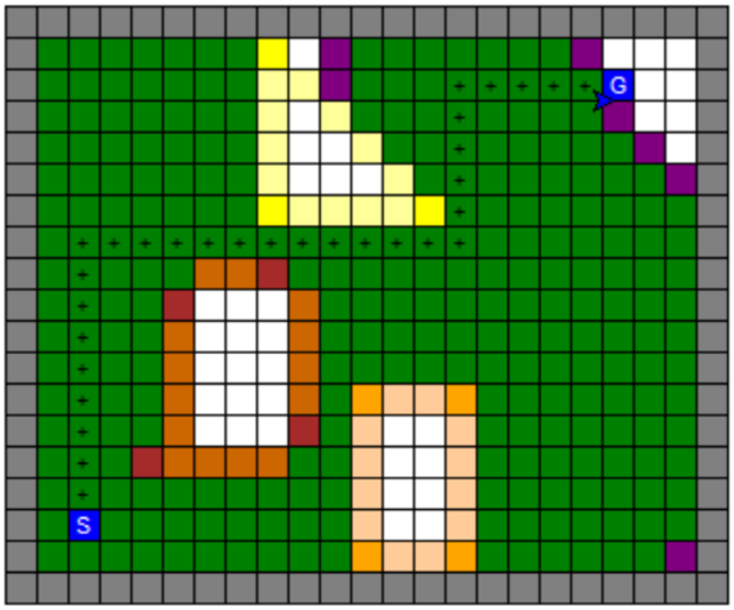
In loop:

- If frontier is empty, then return False.
- Pop head of frontier (use popMinEvaluation() for BFS) and add it to explored (and fill color it).
- For each action can have from node -> create child node, if child node not in explored and frontier -> check whether it is goal, if child node is goal, then reverse traverse (by parent) to find path from source to goal and return True, else add child node to frontier (and fill color it).

b) *Example:*

BFS: exhaust all nodes until the goal is found.

Cost = 31.



c) <u>Conclusion:</u>

- Pros:
    - Complete if branch is finite.
- Cons:
    - Waste of time and space.
    - Not optimal (because exhaust all nodes).

2. *Uniform-cost search:*

a) <u>Idea:</u> expands the node n with the lowest path cost g(n) (equivalent to Dijkstra's algorithm). A replace is added in case a better path is found to a node currently on the frontier.

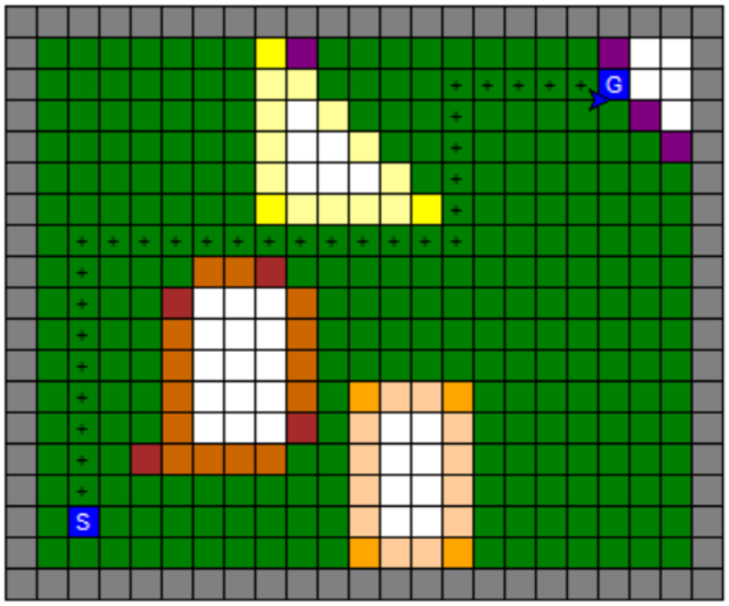
Same as BFS, but:

- Use path cost (g(n)) to pop node and replace node in frontier.
- Only check goal state at after pop node in frontier which have minimum path cost (use popMinEvaluation() for UCS), after if node isn't goal, add it to frontier (same as BFS).
- Not check goal state of initial node and child node in loop.
- In loop, after check child node not in explored and frontier -> check whether in frontier have node (which have same state with child node) which have higher path cost (use nodeHaveInFrontierWithHigherEvaluation() for UCS), if it have, then replace it with child node (use replaceNodeWithHigherEvaluation() for UCS).

b) <u>Example:</u>

UCS with cost always equal -> identical BFS: exhaust all nodes until the goal is found.

Cost = 31.



c) *Conclusion:*

- Pros:
  - Find path which have minimum path cost.
  - Optimal (because nodes expanded in increasing order of g(n)) and complete.
- Cons:
  - Time and space complexity is still exponential.
  - Not optimal (case: cost always equal 1, it will become BFS).

3. *Iterative deepening depth-first search:*

a) *Idea: General strategy, often used in combination with depth-first tree search to find the best depth limit. Gradually increasing the limit until a goal is found. The depth limit reaches the depth d of the shallowest goal node.*

b) *Example: (have code but run failure because stack overflow).*

c) *Conclusion:*

- Pros:
  - Time complexity is lower than DFS, DLF.
  - Space complexity is low.
  - Complete.
- Cons:
  - No optimal in general.

4. Greedy best-first search:

a) <u>Idea:</u> Try to expand the node that is closest to the goal (have minimum heuristic), on the grounds that this is likely to lead to a solution quickly.

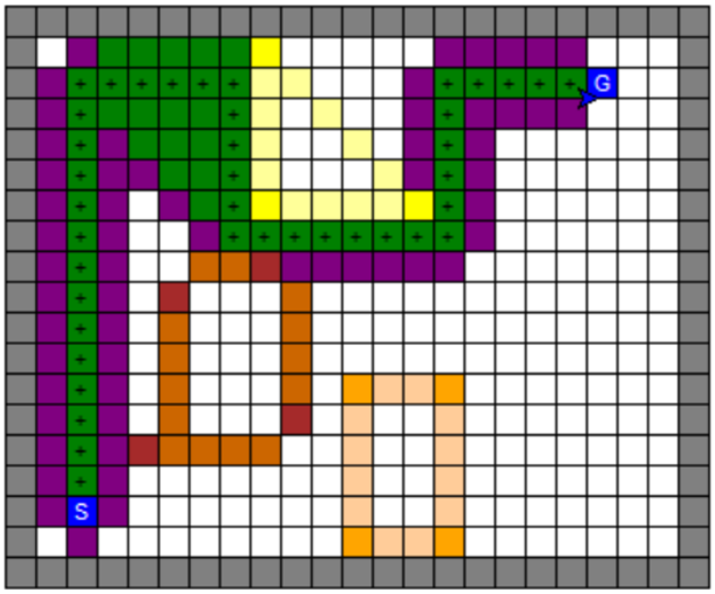
Same as UCS but:

- Use heuristic (h(n)) instead of path cost.
- When pop node in frontier, pop node have minimum heuristic (use popMinEvaluation() for GBFS).
- Don't replace node in frontier same as in UCS.

b) <u>Example:</u>

GBFS: not exhaust all nodes, but at yellow obstacles, because it's blocked, so GBFS exhaust the surrounding nodes.

Cost = 41.



c) <u>Conclusion:</u>

- Pros:
  - Time is good (if good heuristic).
  - Complete.
- Cons:
  - Time is bad (if bad heuristic).
  - Space complexity is still exponential.
  - Not optimal.

5. *A* search:*

a) *<u>Idea:</u> Use heuristic and path cost to guide search and avoid expanding paths that are already expensive.*

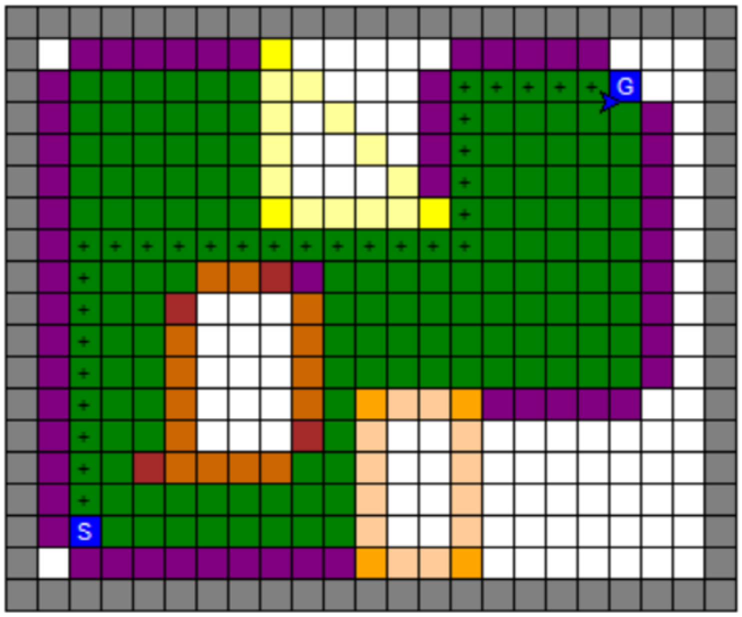
Same as UCS but:

- Use evaluation (f(n) = g(n) + h(n)) instead of path cost.
- When pop node in frontier, pop node have minimum evaluation (use popMinEvaluation() for ASS).
- Check whether in frontier have node (which have same state with child node) which have higher evaluation (use nodeHaveInFrontierWithHigherEvaluation() for ASS), if it have, then replace it with child node (use replaceNodeWithHigherEvaluation() for ASS).

b) *<u>Example:</u>*

ASS: exhaust many nodes because, cost for forward always equal 1.

Cost = 31.



c) *<u>Conclusion:</u>*

- Pros:
  - Time is good (if good heuristic).
  - Complete and optimal.
- Cons:
  - Time is bad (if bad heuristic).
  - Space complexity is still exponential.

V. REFERENCE:

[1] Slides of teacher Bui Tien Len

https://drive.google.com/drive/folders/1CLrbykFrmt8pG_NzleBZTG2bW74zq0iL

[2] AIMA

http://aima.cs.berkeley.edu/