CAP4630 Project 2 – Path Finder in Grid World

Due Date: Friday, 2/24/2023 11:59 PM

In this individual Python project, you are going to implement a path finding agent traveling from point to point in a two-dimensional grid world using some of the search algorithms learned in our class. These search algorithms that you will implement are **breadth-first search (BFS), depth-first search (DFS), greedy best-first search (GBFS), and A* search**. And they are going to be the **graph search versions**. In this project, the environment is the grid world, presented by a grid with certain number of row lines and column lines, where we call each intersection a point. The agent will start at a source point to find a path through other points to reach the destination point. At any point and time, the agent only is allowed to act along the lines in one of four directions: **up, right, down, and left**.

In the grid world, there are two types of obstacles: **enclosures and turfs**. Enclosures are represented as polygons with black points and edges and they are the part of the world that the agent cannot go, not even touching the edges. Turfs also are represented as polygons but with green points and edges. Turfs are places the agent can go through or on edge, but they slow the agent down. Here is the action cost function:

$$ACTION - COST(p, a, p') = \begin{cases} 1, if\ p' is\ ouside\ of\ all\ obstacles \\ 1.5, if\ p' is\ insider\ of\ on\ edge\ of\ a\ turf \end{cases}$$

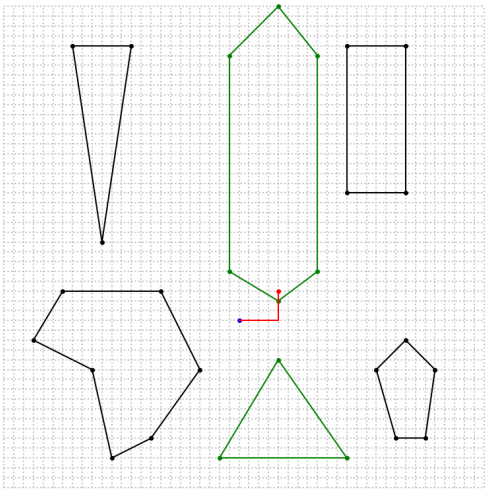
where p is a point, a is an action, and p' is the next point from p doing a. So the action cost function only depends on where p' is. Note BFS and DFS do not consider action costs and path cost for them simply are the number of actions in the path.

The source point is colored blue and destination point red. The path the agent finds using your search algorithms will be colored red as well. You can assume the source and destination points always are outside enclosures.

CAP4630 Project 2 – Path Finder in Grid World

Example Python programs are provided for you to get started familiarizing with the grid world. Inside the search.zip on Canvas, you will find:

1. grid.py file: the Point class, and functions used to draw the grid world.
2. search.py file: function to draw the polygons, and the main function that reads two separate files to build the enclosure polygons and turf polygons, one file for each type of polygons. Your search functions should be called here.
3. utils.py file: the Stack, Queue, and Priority Queue classes.
4. TestingGrid directory: the two world files together define the grid world with enclosure polygons and turf polygons. In each file, every line lists all points in a polygon to represent the polygon. They start with a starting point, and go clockwise to list all other points to define the corresponding polygons. Points are list as x,y coordinate pairs separated semi-colons. Every polygon could be either convex or concave. I provide one testing grid world for you.
5. summary.txt: provide the path costs and numbers of nodes expanded for all four search algorithms using the testing grid in TestingGrid, source point at (8,10), and destination at (43,45).
6. exampleplot.png: an example plot produce using matplotlib library and it can be plotted by running `python3.11 search.py` directly on command line. That said, matplotlib needs to be installed in your Python. This example plot is below, where you see the two types of polygons, the blue source point, the red destination point, and the red path connecting them. This path in the image has a path cost of 8.



Implementation Requirements:

1. Your submission should contain at least grid.py, search.py and utils.py source files, the TestingGrid directory where I will put the testing grid worlds during grading, a README file that describe how to run your program, and a summary.txt file to summarize some.

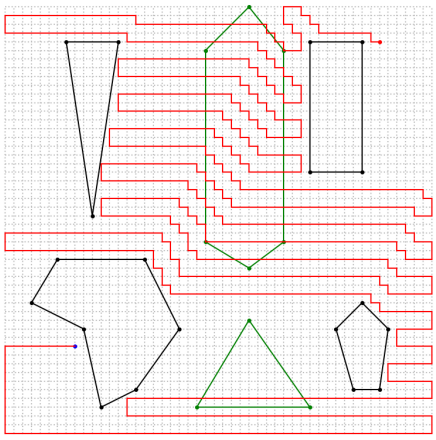
I will use Python 3.11 to run your submissions, and for this I will run command `**python3.11 search.py**` in your project root directory. If there is any package that your program installs and imports, make sure to include such information in the README. If you created and used virtual environment, it should be included in the submission as well.

2. In the TestingGrid directory, create **at least one set** of your own enclosure and turf polygons of your choosing. Be creative. Save results as images like above in this same directory to showcase what your programs can do. Again, be minded that polygons can be **convex or concave**.
3. Feel free to use and change the Python programs provided to you. Feel free to add more Python files per your needs.
4. You may assume the grid world dimension always is 50x50. All point coordinates are correctly provided in the world files, and no two polygons in the whole grid world overlap.
5. You may assume that source and destination points always are outside enclosures.
6. When expanding a node, the order of children nodes to consider always is up, right, down, and left.
7. The four searches should be implemented following the **pseudocodes in textbook**, and the heuristic at a point for both GBFS and A* should be the **straight-line distance** between that point and destination.
8. After completion, the entire project should be zipped and the zip file should be submitted onto Canvas by the due time. **Late project policy** is as detailed in syllabus.
9. Submissions will be checked for peer similarity for **academic integrity**. In case of violation, the minimum penalty will be a zero on the project and a misconduct report.

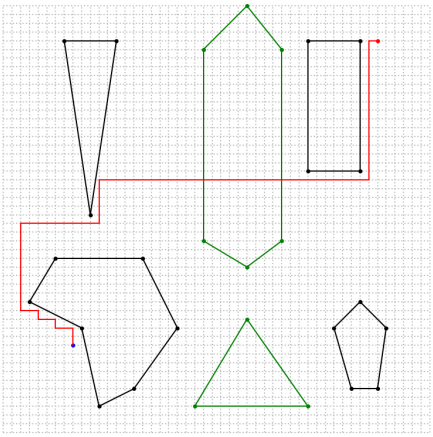
Expected Results:

For the provided testing grid world example, given source at (8,10) and destination at (43,45), your program should compute the paths as follows:

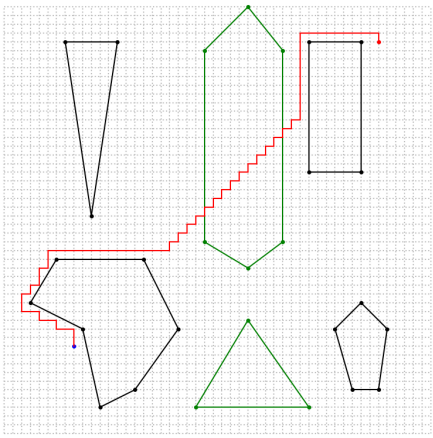
1. DFS:



2. BFS:



3. GBFS (with SLD heuristic):



4. A* (with SLD heuristic):