DS2020 - Artificial Intelligence
Lab 2

Due on 16/02/2026 11.59pm

Instructions:

- Upload to your Moodle account one zip file containing the following.
- You are expected to follow the honor code of the course while doing this homework.
- **This lab should be completed individually**.
- **Rename the Python file** using your roll number (e.g., `12345678.py`) before submitting it to Moodle.

Yantra Hunt with Cost-Based Movement

In this assignment, you will extend the Yantra Hunt problem from Lab 1 by introducing movement costs. Instead of walkable empty spaces represented by '.', each cell now has a cost (integer) associated with moving to that cell. Your task is to implement Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), and A-star Search to find the optimal path for collecting yantras and reaching the exit.

An example grid world is illustrated below:

P	50	#	55	Y2
50	#	55	10	20
20	0	Y1	32	10
#	50	0	#	100
25	50	50	30	E

As in the previous task, the player starts at position 'P' and must collect 'Y1', after which the location of the next yantra is revealed. After collecting the final yantra, the exit 'E' location is revealed. Note that we have values at every walkable position which represent the cost of moving from the previous location to the current location. Additionally, traps (T) have a very large cost(99999) and must be avoided.

Grid Representation The grid consists of the following elements:

- **P**: Player's starting position.

- **Y1**, **Y2**, ... : Yantras to be collected sequentially (only the first yantra is known initially).
- **#**: Walls that cannot be crossed.
- **T**: Traps to be avoided by the player. The cost of moving into a trap is infinite.
- **Numbers**: The cost of moving from the previous location to the current location. For example, if the player moves from '(0, 0)' to '(0, 1)', the cost of this move is the value at '(0, 1)'.
- **?**: Hidden locations (yantras or exit to be revealed).
- **E**: Exit point (revealed after collecting all yantras).
- **Yantras ('Y1', 'Y2', ...) and Exit ('E') have a constant movement cost of 0.**

The accompanying Python file includes the starter code for the lab. The constructor of the `YantraCollector` class processes the input grid and stores the details using appropriate variables. The constructor also automatically creates the sequence in which the yantras must be collected. Helper functions for finding the positions, the yantras, revealing the location of the next yantra or exit are also provided. Do not edit any of these functions.

You are required to implement the following functions in the `YantraCollector` class:

- `get_neighbors(self, position)`: the child generator function that returns the list of adjacent positions reachable from the current position. The reachability of the adjacent positions are checked in the North, East, South and West order.
- `goal_test(self, position)`: returns `True` if the goal condition is satisfied in the current position, else `False`.
- `ucs(self, start, goal)`: finds the path from the start position to the goal position using the Uniform Cost Search algorithm. The function should return the tuple: `(path, frontier_count, explored_count, path_cost)`
  - `path`: List of tuples representing the path from start to goal.
  - `frontier_count`: Number of nodes in the frontier list at the time of exiting the function.

  - `explored_count`: Number of nodes in the explored list at the time of exiting the function.
  - `path_cost`: Total cost of the path.

- `heuristic(self, position, goal)`: Define a heuristic function to improve search efficiency. The heuristic does not need to be admissible, but you **cannot** use Euclidean or Manhattan distance or any other standard distance metric. The function returns the estimated cost to the goal.

- `gbfs(self, start, goal)`: finds the path from the start position to the goal position using the Greedy Best-First Search algorithm. The function should return the tuple: `(path, frontier_count, explored_count, path_cost)`
  - `path`: List of tuples representing the path from start to goal.
  - `frontier_count`: Number of nodes in the frontier list at the time of exiting the function.
  - `explored_count`: Number of nodes in the explored list at the time of exiting the function.
  - `path_cost`: Total cost of the path.

- `a_star(self, start, goal)`: performs A* Search to reach the goal. The function should return the tuple: `(path, frontier_count, explored_count, path_cost)`
  - `path`: List of tuples representing the path from start to goal.
  - `frontier_count`: Number of nodes in the frontier list at the time of exiting the function.
  - `explored_count`: Number of nodes in the explored list at the time of exiting the function.
  - `path_cost`: Total cost of the path.

- `solve(self, strategy)`: complete this function to call functions `ucs()`, `gbfs()`, `a_star()` functions appropriately and consecutively for completing the yantra hunt. Ensure to combine the path from the different consecutive calls, as well the total number of nodes in the frontier and explored list for the entire search. The function should return the tuple: `(full_path, total_frontier_nodes, total_explored_nodes, total_cost)`
  - `full_path`: List of tuples representing the full path to collect all yantras and exit.

- `total_frontier_nodes`: Sum of frontier_count across all searches.
- `total_explored_nodes`: Sum of explored_count across all searches.
- `total_cost`: Total cost of the path.

Your code should work with any grid size, at least one yantra, and arbitrary yantra pickup sequence