COM S 476/576 Project 1: Discrete Planning

Due Feb 2 at 11:59pm

Consider a robot moving on a 2D grid. Each grid point has an integer coordinate of the form (i, j) where $i \in \{0, \ldots, M\}$, $j \in \{0, \ldots, N\}$, and M and N are positive integers. The state space is, therefore, defined as

$$X = \{(i, j) \in \mathbb{Z} \times \mathbb{Z} \mid 0 \leq i \leq M, 0 \leq j \leq N\}. \tag{1}$$

The configuration of the grid is given by a 2D array of the form

$$G = \begin{bmatrix} O_{0,0} & O_{1,0} & \cdots & O_{M,0} \\ O_{0,1} & O_{1,1} & \cdots & O_{M,1} \\ \vdots & \vdots & \vdots & \vdots \\ O_{0,N} & O_{1,N} & \cdots & O_{M,N} \end{bmatrix}, \tag{2}$$

where $O_{i,j} \in \{0, 1\}$ indicates whether cell (i, j) is occupied, i.e., $O_{i,j} = 1$ if and only if cell (i, j) is occupied by an obstacle. Notice that when writing G as a 2-dimensional list, its indices are such that $G[i][j]$ corresponds to cell (j, i).

The robot takes discrete steps in one of the four directions (up, down, left, right) with the corresponding actions $u_{up} = (0, 1)$, $u_{down} = (0, -1)$, $u_{left} = (-1, 0)$, and $u_{right} = (1, 0)$. Let $U = \{u_{up}, u_{down}, u_{left}, u_{right}\}$. The action space for $x = (i, j)$ is defined as $U(x) \subseteq U$ such that

- $u_{up} \in U(x)$ if and only if $j < N$ and $O_{i,j+1} = 0$,
- $u_{down} \in U(x)$ if and only if $j > 0$ and $O_{i,j-1} = 0$,
- $u_{left} \in U(x)$ if and only if $i > 0$ and $O_{i-1,j} = 0$,
- $u_{right} \in U(x)$ if and only if $i < M$ and $O_{i+1,j} = 0$.

Task: Implement the 3 variants of forward search algorithm: breadth-first, depth-first, and A*. The problem description will be provided in a json file, containing the following fields:

- `"G"`: a 2-dimensional list representing the grid configuration G,
- `"xI"`: a list $[i, j]$ specifying the initial cell $x_I = (i, j) \in X$, and
- `"XG"`: a list of $[i, j]$'s, each corresponding to a goal cell $x_G \in X_G \subset X$.

Your code should take 2 inputs: (1) the algorithm (`bfs`, `dfs`, or `astar`), and (2) the path to the json file, which specifies the problem description with the above format. It should output a json file, containing the following fields:

- `"visited"`: the list of visited cells, and

- `"path"`: the list of cells specifying the path from x_I to X_G.

For example, if your code is `project1.py`, running

```
python project1.py project1_desc.json --alg bfs --out project1_bfs.json
```

should output `project1_bfs.json`, which contains the output of running breadth-first search, including `"visited"` and `"path"` for the problem described in `project1_desc.json`. Example of `project1_bfs.json` and `project1_desc.json` can be found on the course github repo.

Requirements:

- To simplify grading, please explore the actions in the following order: u_{up}, u_{down}, u_{left}, u_{right}.
- Do NOT implement each algorithm as a separate function. Instead, you should implement the general template for forward search (See Figure 2.4 in the textbook). Then, implement the corresponding priority queue for each algorithm.

  Hint: The only difference between different algorithms is how an element is inserted into the queue. So at the minimum, you should have the following classes:

  - `Queue`: a base class for maintaining a queue, with `pop()` function that removes and returns the first element in the queue. You may also want this class to maintain the parent of each element that has been inserted into the queue so that you trace back the parent when computing the path from x_I to a goal cell $x_G \in X_G$.
  - `QueueBFS`, `QueueDFS`, `QueueAstar`: classes derived from `Queue` and implement `insert(x, parent)` function for inserting an element `x` with parent `parent` into the queue.

  With the above classes, you can implement a general `fsearch(G, U, xI, XG, Q)` function, which takes as input the grid configuration `G`, the list `U` of actions, the initial cell `xI`, the list `XG` of goal cells, and the queue object `Q` that contains `insert` and `pop` functions (and possibly some other functions, e.g., for computing a path).

- Following the previous bullet, there should be only one conditional statement for the selected algorithm (BFS, DFS or Astar) in the entire program.
- For A*, assume that the cost of each action is 1, i.e., cost-to-come to cell x' is given by $C(x') = C(x) + 1$ where x is the parent of x'. Suppose a state is written as $x = (x^1, x^2)$. Use $\hat{G}^*(x) = \min_{x_G \in X_G} \left(|x^1 - x_G^1| + |x^2 - x_G^2| \right)$ as the underestimate of the optimal cost-to-go at x.
- Please feel free to use external libraries, e.g., for heap, queue, stack or implement them yourself.

Submission: Please submit a single zip file on Canvas containing the followings:

- your code (with comments, explaining clearly what each function/class is doing), and
- a text file explaining clearly how to compile and run your code.