CS 747: Programming Assignment 2

Name: Ansh Khurana
Roll Number 170050035

Assumptions / Design Decisions:

1. MDP format should be as given in the the problem statement. Any deviation causes program to exit
2. For episodic MDPs, the policies have action set to -1 for end_states (since no outgoing transition is possible). (Any state with no outgoing transition has action set to -1 in optimal policy output)

Value Iteration:

1. **epsilon** = 1e-10
2. For value iteration, initial values of the value function are set as: -1e10
3. **vi convergence condition** : `np.linalg.norm(Vnew - Vold) / numStates < epsilon`

Howard Policy Iteration:

1. For obtaining value function from policy, using `np.linalg.solve`
2. For picking improvable action, just pick the first action in the list of valid choices. (actions for which Q_sa > V_s)

LP Solver:

1. Solver: `pulp.PULP_CBC_CMD(msg=False).solve(prob)`
2. Sometimes the output doesn't print in 6 decimal places (I am not limiting precision of values, so they are the default python floating point numbers / LpContinuous variables) but the test cases are still passing and error of < 1e-4 is maintained.

Task 2:

Runtime:

vi ~ 35 seconds, lp ~ 20 seconds, hpi: slow

Please run using **vi** or **lp** options only. hpi is slower since it needs to solve linear equations multiple times.

Maze MDP Design

States:

Every position in the maze has been considered as a state, and thus we can assign row majoring numbering to states. Some states are unreachable as they are walls and thus transition probability to valid (free) state to invalid (wall) state is set to 0 (excluded from MDP transition).

Actions:

A = {0,1,2,3}

actions 0 - North, 1 - East, 2 - South, 3 - West

And correspond to displacement of the agent:

```
action[0] = (-1, 0)
action[1] = (0, 1)
action[2] = (1, 0)
action[3] = (0, -1)
```

Transition:

Transitions has been set from all valid states to adjacent states with actions N,E,S,W whenever possible with prob=1.0 since the action in a particular direction returns the next state deterministically. (transition to wall has prob = 0)

T(s,a,s') = 1.0 (if s and s' are both valid/free states, and we reach s' from s by taking moving in direction specified by a)

T(s,a,s') = 0.0 (if any of s,s' are invalid/wall states) (omitted from the MDP output of encoder.py)

Reward: R(s, a, s')

The reward function depends only on s' in my model. That is, the reward obtained in each step is based on which state you move to.

if s' = valid_state, R=-2 (to penalize longer paths and return the shortest)

if s' = end_state, R=10000 (to award reaching finish)

if s' = start_state, R=-10 (should not go back to start)

if s' = wall_state, R=-1000 (should not go hit a wall) (this would never happen since p=0 for such a transition)

Gamma: 0.9

Justification for formulation:

The value (expected reward) obtained from start_state would be max for the shortest path from start to state to end state since:

for every move, we give a reward of -2 and for reaching the end state we get +100000 reward.

So any path longer than the shortest path would have

(i) more negative step terms (more -2 reward terms in addition to the ones we obtain from the shortest path even if they would have smaller values due to gamma, but still lead to a more negative value as it is being added as additional steps compared to the shortest one).

(ii) the final positive reward we get would also be diminished by larger factor (gamma^number_of_steps)

Thus, any path other than the shortest path has a lower expected reward, and thus the optimal policy should direct us to get to the shortest path.