Assignment 2

1 TSP

The traveling salesman problem (TSP) is to find the shortest possible route that allows visit each city exactly once and return to the starting point. When it formulated as a graph search problem, it has a collection of nodes connected by edges, where each edge has a cost associated with it. The detailed formulation is as follows:

- **States:** A list of nodes that have been visited once (starting node visited twice)
- **Initial States:** Start from a node (random or pre-planned), where other nodes have not been visited.
- **Operators:** Each edge has a cost function, so the cost operator is the cost of edge between 2 nodes (happen where moving from a node to another).
- **Goal Test:** All nodes have been visited exactly once and it ends on the starting point (The traveler has returned to the starting node).

The problem becomes one of finding the optimal path that satisfies the goal condition while minimizing the total cost.

2 CSP

A constraint satisfaction problem (CSP) is a type of problem where we are finding values of variables while obeying certain constraints. The problem of constructing a crossword puzzle can be formulated as a CSP:

- **Variables:** Each row r_i and column c_j in the grid is a separate variable.
- **Domains of Possible Values:** The domain for each variable is the list of words from the dictionary.
- **Constrains:**
    - For each intersection between a row r_i and a column c_j, we define a condition such that the row and column intersect at specific positions. For example, if they intersect at the mth position in the row and the nth position in the column, then the constrain ensures that $r_i[m] = c_j[n]$, meaning that the letter at those positions must be the same.
    - If the row and column do not intersect, the positions m and n are set to 0, meaning no constraint is applied there.

3 MDP

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. This investment problem can be formulated as a MDP, where each state represents the amount of money after each year.

- **States:**
  - Each state is represented by a tuple $(x_1, x_2, \ldots, x_k)$, where $k \leq 3$ indicates the number of actions taken so far, and each x_i is the return from an investment action.
  - The possible values of x_i are 1.1 (for CD investment), 1.3 (for a successful stock investment) and 0.9 (for a losing stock investment).
  - The initial state is () (no actions taken), and we reach an end state once three actions have been performed.
- **Transition Model:**
  - If an investment in a CD is chosen, the next state is $s.append(1.1)$, and the probability is 1.
  - If stocks are chosen, the probability of moving to $s.append(1.3)$ (successful investment) is 0.7, while the probability of moving to $s.append(0.9)$ (unsuccessful investment) is 0.3.
  - After 3 actions, the proves moves to the end state.
- **Rewards:**
  - If the final amount of money $M(s)$ in a state is greater than or equal to 1 (i.e. no loss of principal), the reward is 0.
  - Otherwise, if the amount is less than 1, the reward is -10.
  - For intermediate states, the reward is the difference between the money in the new state and the previous state, $Reward = M(s') - M(s)$.
- **Optimal Policy:**
  - The optimal policy should be ensured that stocks are chosen only if a 10% loss can be recovered through future investments in CDs.
  - Essentially, the policy dictates choosing stocks if there are enough remaining years to make up for a potential loss by switching to CDs.

4 Boundary Following Robot

4.1 MDP

- **States:**
  - States: Assume there are 8 states of input sensors, represented as $S = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]$, which are top, bottom, left, right, top left, top right, bottom left, bottom right proximity.
  - Action: Move north (up), move south (down), move west (left), go east (right).
  - Terminate: The robot is running, unless blocked state is triggered $S = [1,1,1,1,1,1,1,1]$
- **Transition Model:**
  - If the current move is legal, the robot will move, and the state will be updated to its newest position.
  - If the current move is illegal, the state will remain unchanged.
- **Reward:**
  - Let the reward variables be x_1, x_2, x_3, x_4, where $x_1 = s_1 s_2, x_2 = s_3 s_4$, $x_3 = s_5 s_6, x_4 = s_7 s_8$.
  - If $x_1 > 0$, the reward for moving north is 1, otherwise -1
  - If $x_2 > 0$, the reward for moving west is 1, otherwise -1
  - If $x_3 > 0$, the reward for moving south is 1, otherwise -1
  - If $x_4 > 0$, the reward for moving east is 1, otherwise -1
  - Otherwise, the robot move north, the reward is 0.
- **Policy:**
  - Move north, if $x_4 x_1$
  - Move west, if $x_3 x_4$
  - Move south, if $x_2 x_3$
  - Move east, if $x_1 x_2$
  - Otherwise, move north

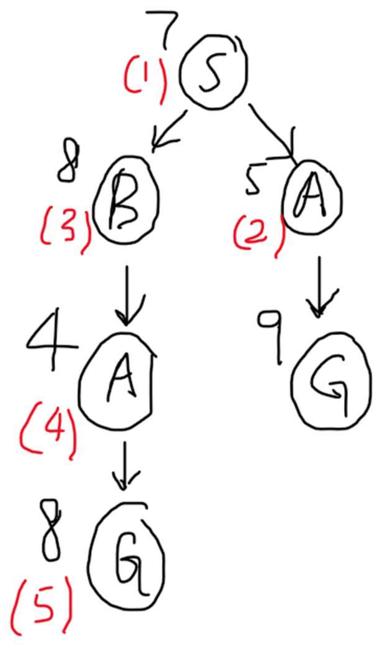
4.2 Reinforcement Learning (RL)

- **States:**
  - Input sensors state $S = [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]$
  - Terminate state $S = [1,1,1,1,1,1,1,1]$
  - Action: Move north (up), move south (down), move west (left), go east (right)
- **Policy:**
  - The policy $\pi(s)$ can choose the action $a \in Actions$ that maximize $Q(s, a)$ with a probability of $1 - \epsilon$, and with ϵ probability, a random action is chosen from the set of possible actions.

- **Reward:**
  - $x_1 = s_1 s_2, x_2 = s_3 s_4, x_3 = s_5 s_6, x_4 = s_7 s_8.$
  - If $x_1 > 0$, the reward for moving north is 1, otherwise -1
  - If $x_2 > 0$, the reward for moving west is 1, otherwise -1
  - If $x_3 > 0$, the reward for moving south is 1, otherwise -1
  - If $x_4 > 0$, the reward for moving east is 1, otherwise -1
  - Otherwise, the robot moves north, the reward is 0.
- **Calculations:**
  - Learning process is Monte Carlo Method, recording a sequence of states, actions and rewards $D = [s_1, a_1, r_1, s_2, a_2, r_2, \ldots]$
  - Utility at state s_i is calculated by $u_i = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \cdots$, where γ is the discount factor.
  - Q-value is estimated as the average of all utilities of the state-action pairs $Q(s, a) = average\ of\ \{u_i \mid s_i = s, a_i = a\}$.

5 A* Search

A* search is used to find the shortest path from a starting point to a goal in a graph. By implementing A* search by tree in this problem, the solution is:



6 Minimax Algorithm

The minimax algorithm is used in decision-making and game theory. By applying the minimax algorithm to compute the optimal decision at each level, the values of the nodes are given as:

- $I = 0, F = 5, G = 8, C = 4, B = 3, A = 4$

7 Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique for the minimax algorithm, it can be achieved by reduce the number of nodes that are evaluated in the search tree, making the decision process faster. For this case:

- **Left to right:** Node N and L pruned.
- **Right to left:** Node K and I pruned.

8 Programming

Saved in a separate zip file, in folder "pacman", files edited:

- search.py
- searchAgents.py

To run (cd to pacman folder):

- Task 1:
  - python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
- Task 2:
  - python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
  - python pacman-hw2/pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
- Task 3:
  - python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5