CSE 415 Winter 2021 Assignment 4

Last name: Lin First name: Jia-You (Billy)

Due Wednesday night February 10 via Gradescope at 11:59 PM. You may turn in either of the following types of PDFs: (1) Scans of these pages that include your answers (handwriting is OK, if it's clear), or (2) Documents you create with the answers, saved as PDFs. When you upload to GradeScope, you'll be prompted to identify where in your document your answer to each question lies.

Do the following five exercises. These are intended to take 20-25 minutes each if you know how to do them. Each is worth 20 points.

---

1 Blind Search

The game of Pentago is a more complicated version of the standard Tic-Tac-Toe. In this game, there are four standard Tic-Tac-Toe boards arranged in a 2 by 2 grid. The objective of this game is to get five of your "color" (X or O) in a row. However, the twist in this game is that upon the placement of your piece on the board, you must also rotate one of the standard Tic-Tac-Toe boards 90 degrees clockwise or counterclockwise. This is illustrated below.

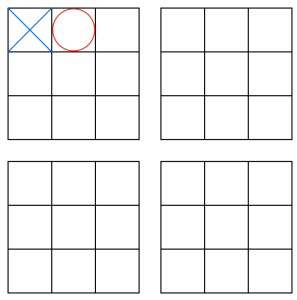


Figure 1: Initial State

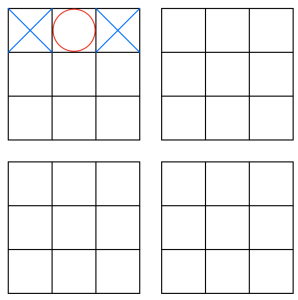


Figure 2: Place X

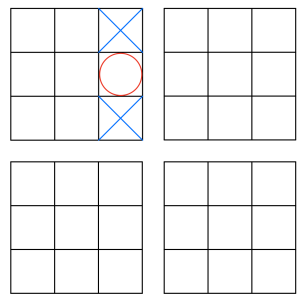


Figure 3: Rotate Board

Suppose for this problem you are only allowed to turn a board clockwise.

(a) (5 points) Design a state representation of the Pentago board. You can express this in Python, English, or pseudocode.

A state has 4 fields for 4 quadrants : q_1, q_2, q_3, q_4.
Each of them is a 2D array.
"X is represented by 1 and O is represented by 0.

for example above

$q_1 = [[1, 0], [], []]$
$q_2 = [[], [], []]$
$q_3 = [[], [], []]$
$q_4 = [[], [], []]$

(b) (10 points) Describe a function **successors(s, player)** where s represents the state of a board and player represents whose turn it is. For this problem, assume you have some predefined function **rotate90(arr)** that gives you the 90 degree clockwise rotation of a 3 by 3 matrix. You may use Python, English, or pseudocode.

We start with an empty list of successors
Then we iterate through every index of the
2D arrays (q1,q2,q3,q4) in the current state.
whenever we find an empty spot (len of urrent
row is <3), we make a deep opy of the state
and fill in the new state's empty spot, Having
this new state, we further go into a for loop,
in which we make copies of the new state, rotate
each quadrant in different iterations, and append
the rotated new states into the successor list.
Then we move on to the next index. Finally, we return
the list of successors.

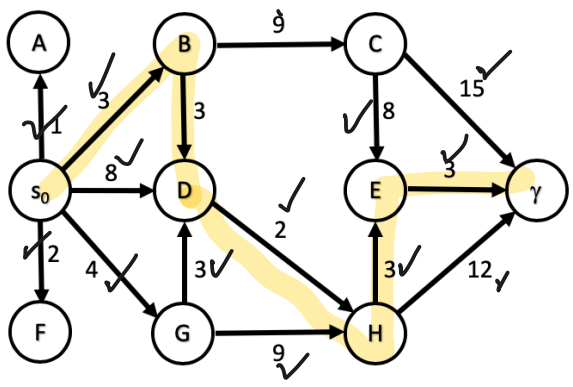
(c) (5 points) Describe a function **isGoal(s, player)** where **s** is the state of the board and **player** is the person for whom we are checking is in a goal state. You may use Python, English, or pseudocode.

We keep 3 counts : vertical_count, horizontal_count,
and. diagonal count, and we iterate through each element
in the 2D arrays of the state. We increment 1 to vertical_count
if the same columns of different row have the same value, reset
to 0 once we find an unmatch. We do the same for the horizontal_count
by checking columns in the same rows. To count the diagonal,
we look at the elements that have the same indices for columns
and rows. Return true. once one of the counts == 5.
(counting all in same for loop might be complicated.
breaking into 3 for loops would simplify the iterations and counting)

2 Heuristic Search

(a) (5 points) Consider the following statement: *The best heuristic is always the one that gives you the estimate closest to the true cost.* Explain why you agree or disagree with the statement.

I disagree with the statement because we need to verify if the heuristic function is admissible.



O : violation in consistency
X : violation in admissibility

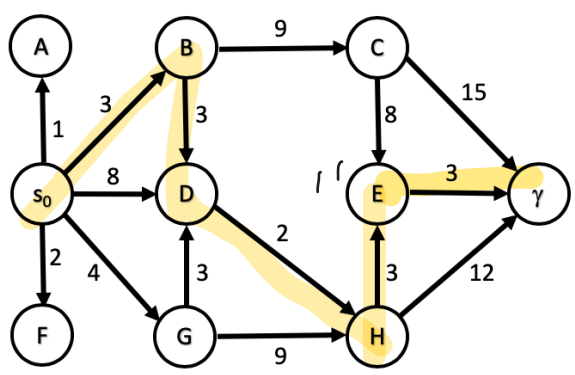
state (s)	s_0	A	B	C	D	E	F	G	H	γ
heuristic $h_1(s)$	14	15	7	8	6	3	20	10	4	0
heuristic $h_2(s)$	14	16	10	10	7	2	16	14	5	0
heuristic $h_3(s)$	14	15	11	10	8	3	16	10	6	0

(b) (5 points) Which heuristics (h_1, h_2, h_3) shown above are admissible? h_1, h_3.

(c) (5 points) Which heuristics (h_1, h_2, h_3) shown above are consistent? h_3

(d) (5 points) Which of the 3 heuristics shown above would you select as the best heuristic to use with A* search, and why? Refer to consistency/admissibility in your justification.

I will choose h_3 because h_3 is both admissible and consistent because it will find the shortest path as soon as reaching to the goal state (admissible) and will never have to reexpand a node (consistent).



state (s)	s_0	A	B	C	D	E	F	G	H	γ
heuristic $h_4(s)$	14	28	11	11	8	3	24	11	6	0

(e) (5 points) Referring back to the graph again, trace out the path that would be followed in an A* search, given the heuristics provided above. As you trace the path, complete the table below, indicating which nodes are on the open and closed lists, along with their 'f' values:

	Open	Closed
Starting A^* search	$[s_0, 14]$	empty
s_0	[B,14], [G,15] [D,16], [F,26] [A,29]	[S0,14]
B	[D,14], [G,15], [C,23], [F,26] [A,29]	[S0,14], [B,14]
D	[H,14], [G,15], [C,23], [F,26] [A,29]	[S0,14], [B,14], [D,14]
H	[E,14], [G,15], [γ,20], [C,23] [F,26], [A,29]	[S0,14], [B,14], [D,14], [H,14]
E	[γ,14], [G,15], [C,23], [F,26] [A,29]	[S0,14], [B,14], [D,14], [H,14] [E,14]
γ	[G,15], [C,23] [F,26] [A,29]	[S0,14], [B,14], [D,14] [H,14] [E,14], [γ,14]

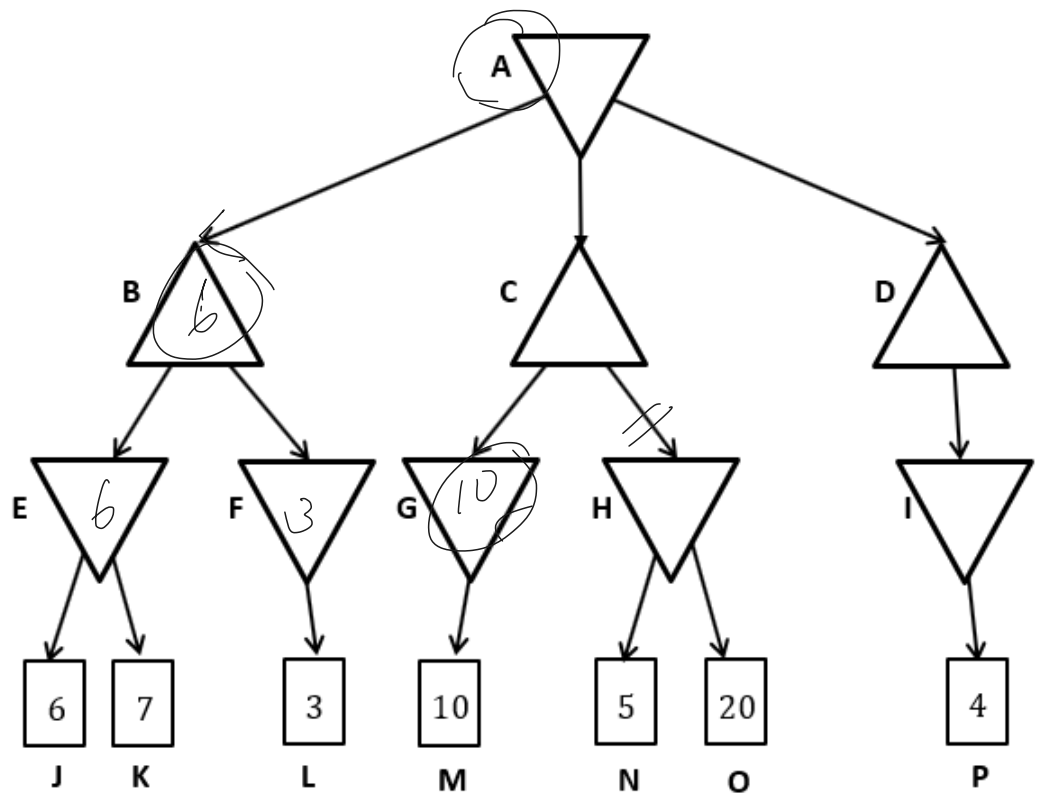
⇒ shortest path: $S_0 \to B \to D \to H \to E \to \gamma$

3 Adversarial Search

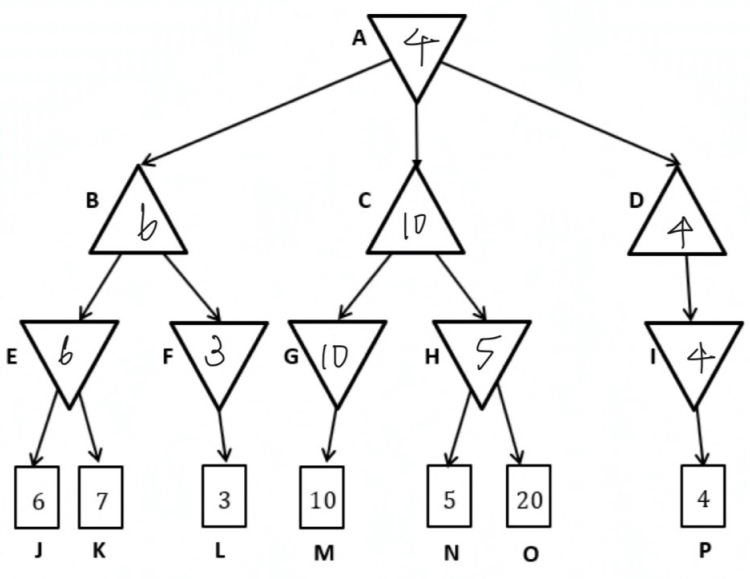
Minimax game-tree search finds a best move under the assumptions that both players play rationally, to either maximize or minimize the value of the same static-evaluation function to a certain ply limit. (It can also do well even when those assumptions are relaxed somewhat.) However, the quality of a move usually improves as the maximum ply is increased. That usually makes minimax take a lot longer. Alpha-beta pruning is a method for speeding up minimax by eliminating any subtrees from the search that can be identified as not able to contribute to the outcome. However, alpha-beta pruning's success is dependent upon the order in which the successors of a state are analyzed.

For each of the four following methods, determine the number of cutoffs, the number of leaf nodes statically evaluated and the total number of states that would have to be generated. Note that this example has a minimizing node at the root; that means we are computing the value of the best move for the minimizing player and the minimizing player's best move, as the overall objective in this problem.

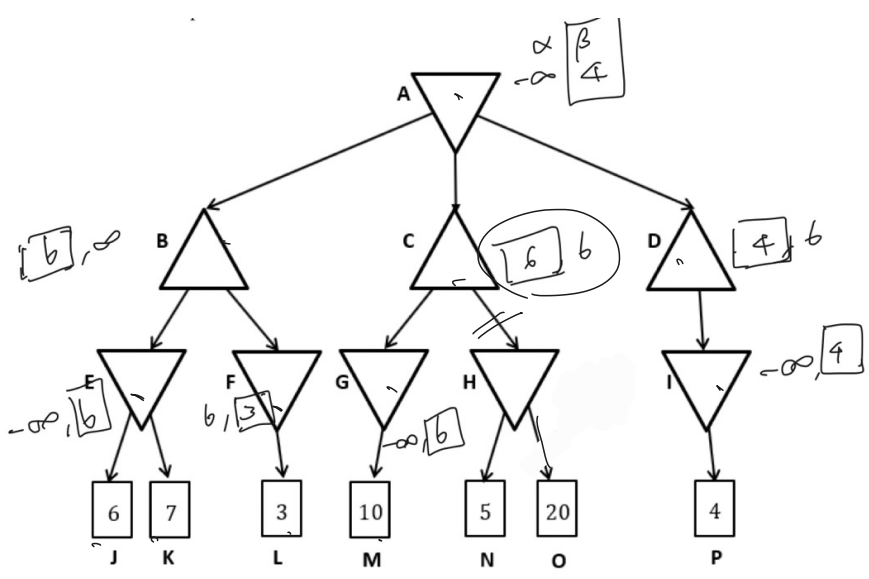
For parts (c) and (d) there are blank tree diagrams you should complete to show the new order in which the space is searched.



minimax



alpha-beta pruning



(a) (5 points) Straight minimax search, depth-first, left-to-right. Show the backed-up values at each internal node. Then fill in the table. Total number of states generated should include the root, and should include those leaf nodes that had to be statically evaluated.

Number of cutoffs:	0
Number of leaf nodes processed:	7
Total number of states generated:	16

(b) (10 points) Alpha-beta pruning, left-to-right, on the given tree. Mark where cutoffs occur on the tree, and fill out the table:

Number of cutoffs:	1
Number of leaf nodes processed:	5
Total number of states generated:	13

(c) (10 points) Alpha-beta pruning, using a secondary evaluation function $f_2(s)$. Rather than going depth-first, use the following method. Start at the root, A; call this S_c for current state. To process S_c, check whether it is a leaf node (i.e., at the maximum ply). If so, return its static value (the normal static evaluation, the values are given in the rectangles in the diagram). Otherwise, S_c is an internal node, so generate all its successors (its immediate children, but not their children, etc.). Apply f_2 to each of the children, and sort them into best-first order. (If S_c is a maximizing node, then highest is best. Else lowest is best.) Process these children in this best-first order, using the regular alpha-beta method, by first calling recursively on the best child, then the next best child (unless a cutoff happens and the rest of the children of S_c can be ignored). Return the best value found among those children not cut off.

For this part, use f_2 as given in this table. Leave nodes J through P in the same relative order as in the original diagram. (In practice, an agent designer might use a single static evaluation function to serve both the usual purpose of evaluating leaf nodes and the new purpose of pre-evaluating internal nodes, but one point of this exercise is to show that they can be different functions, possibly investing more or fewer computational resources into finding a best ordering for the successors of a state prior to alpha-beta pruning.)

Node s:	A	B	C	D	E	F	G	H	I
$f_2(s)$:		7	6	3	4	2	4	5	1

Complete the diagram of the re-ordered tree, labeling each internal node with the letter for the appropriate state in the original diagram. Draw in the missing edges, since the tree's shape may now be a little different. Mark where cutoffs occur on the tree, and fill out the table:

processed: A, D, I, P, C, H, N, O, B

successors of A: [D, 3], [C, 6], [B, 7]

D: [I, 1]

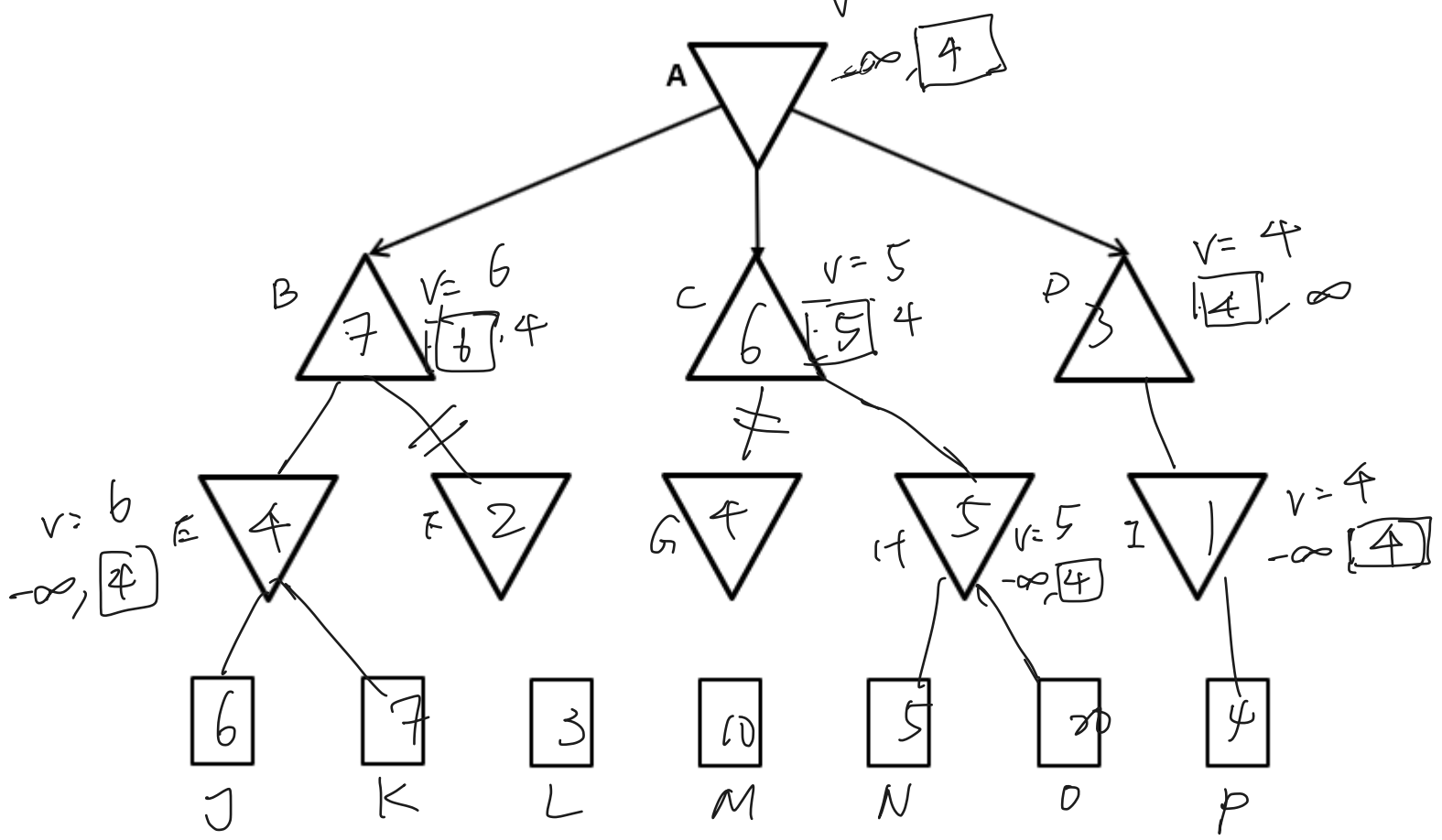
I: [P, 4]

C: [H, 5], [G, 4]

H: [N, 5], [O, ∞]

B: [E, 4] [F, 2]

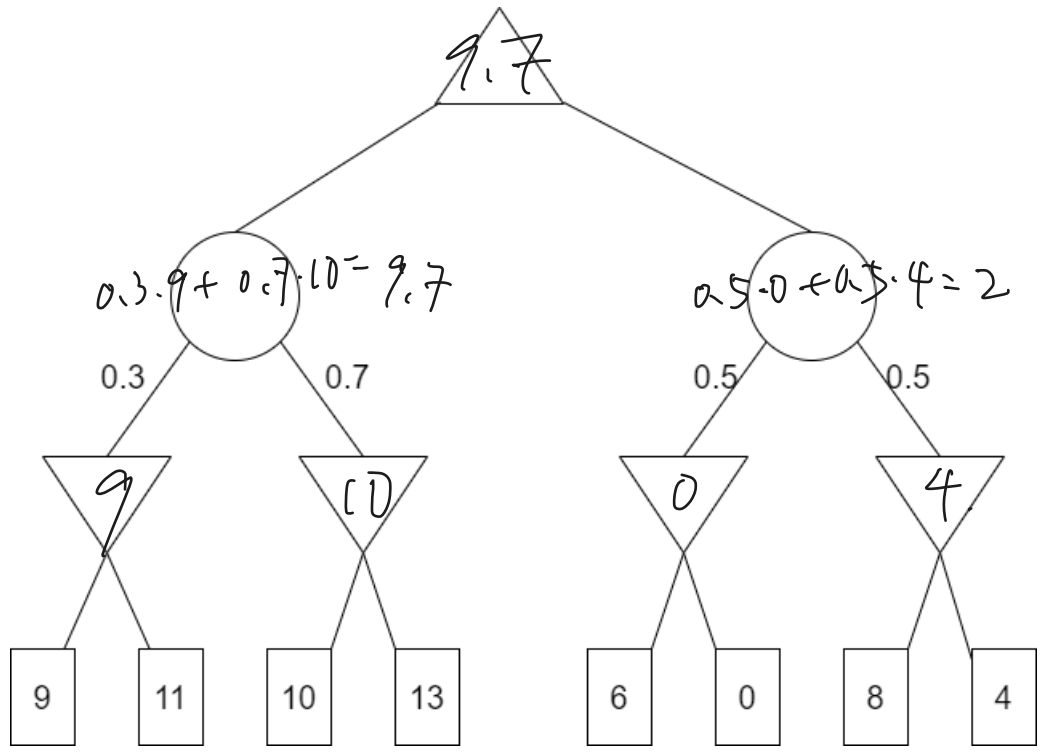
E: [J, 6] [K, 7]



Number of cutoffs:	2
Number of leaf nodes processed:	5
Total number of states generated:	12

Note that the total number of states generated must be sure to include all the children of any internal node that did not get cut off, since we assume that f_2 cannot be applied to them unless they are created.

Consider the following expectimax game tree. Note that the new ◯ nodes represent expectation nodes and the probability of their successors are denoted on the outgoing edges of these nodes.



(d) (5 points) Fill in the nodes in the tree with the correct values selected by the maximizing and minimizing players during the expectimax algorithm.

4 Markov Decision Processes

Consider the following game. You have three coins - one gold and 2 silver, each of which are tossed independently of one another and give heads or tails with equal probability. If you get heads on the gold coin, you get a score of 2, and 0 otherwise. Getting a head on either of the silver coins gives a score of 1 or 0 for heads/tails respectively. On every turn, you flip all 3 coins and record the total score of all the three coins. You keep a running score that adds up the scores produced in each turn.

At any point of time, you can either flip the coins or stop if the running score is less than 6. If the running score reaches or exceeds 6, you "go bust" and go to the final state, accruing zero reward.

When in any state other than the final state, you are allowed to take the stop action. When you stop, you reach the final state and your reward is the running total score if it is less than 6.

Note: there is no direct reward from tossing the coins (or we could say that there is a reward but it's always 0). The only non-zero reward comes from explicitly taking the stop action. Discounting or not should not matter in the MDP for this game, but for the record, we assume no discounting (i.e., $\gamma = 1$).

(a) (6 points) Write down the states (in any order) and actions for this MDP. (Hint: there are 7 states in total and each should correspond to a numeric value except the final states)

$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$.

$A = \{\text{toss}, \text{stop}\}$

state	running score
s_0	0
s_1	1
s_2	2
s_3	3
s_4	4
s_5	5
s_6	6

(b) (10 points) Give the full transition function $T(s, a, s')$. Here s is a current state, a is an action, and s' is a possible next state when a is performed in s. Assuming your states are s_0, s_1, s_2, s_3 etc., and actions are a_0, a_1 etc., some examples of how you should write the function are as follows:

$$\rightsquigarrow T(s, a_0, s') = \langle \text{value} \rangle; s = s_0, s' \in \{s_1, s_2, s_3, \ldots\}$$

$$\rightsquigarrow T(s_0, a_1, s_1) = \langle \text{value} \rangle$$

$T(s, toss, s') = \frac{1}{8}$; $s = s_0$, $s' = \{s_0, s_4\}$
$T(s, toss, s') = \frac{1}{4}$; $s = s_0$, $s' = \{s_1, s_2, s_3\}$.

$T(s, toss, s') = \frac{1}{8}$; $s = s_1$, $s' = \{s_1, s_5\}$
$T(s, toss, s') = \frac{1}{4}$; $s = s_1$, $s' = \{s_2, s_3, s_4\}$.

$T(s, toss, s') = \frac{1}{8}$; $s = s_2$, $s' = \{s_2, s_6\}$
$T(s, toss, s') = \frac{1}{4}$; $s = s_2$, $s' = \{s_3, s_4, s_5\}$.

$T(s, toss, s') = \frac{1}{8}$; $s = s_3$, $s' = s_3$
$T(s, toss, s') = \frac{1}{4}$; $s = s_3$, $s' = \{s_4, s_5\}$
$T(s, toss, s') = \frac{3}{8}$; $s = s_3$, $s' = s_6$.

$T(s, toss, s') = \frac{1}{8}$; $s = s_4$, $s' = s_4$
$T(s, toss, s') = \frac{1}{4}$; $s = s_4$, $s' = s_5$
$T(s, toss, s') = \frac{5}{8}$; $s = s_4$, $s' = s_6$.

$T(s, toss, s') = \frac{1}{8}$; $s = s_5$, $s' = s_5$
$T(s, toss, s') = \frac{7}{8}$; $s = s_5$, $s' = s_6$

$T(s, toss, s') = 1$; $s = s_6$, $s' = s_6$
$T(s, stop, s') = 1$; $s \in \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, $s' = s_6$

(c) (2 points) Give the full reward function $R(s, a, s')$.

$R(s, toss, s') = 0$; $s \in S$, $s' = S$
$R(s_0, stop, s_6) = 0$
$R(s_1, stop, s_6) = 1$
$R(s_2, stop, s_6) = 2$
$R(s_3, stop, s_6) = 3$.
$R(s_4, stop, s_6) = 4$.
$R(s_5, stop, s_6) = 5$
$R(s_6, stop, s_6) = 0$.

(d) (2 points) What is the optimal policy? There is no need to perform value iteration or use any fancy math; just write your answer in words.

$Q(s, toss)$: $0 \cdot \frac{1}{8} + 1 \cdot \frac{1}{4} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{4} + 4 \cdot \frac{1}{8} = 2$

The weighted average of the reward you would get after tossing a coin is 2.
Therefore, if the current sum you have is < 4, you should toss the coin again.
In other words, if you are in s_0, s_1, s_2, s_3, then you should toss again;
if you are in s_4, s_5, then you should stop.

5 Computing MDP State Values and Q-Values

Recently, Jim has been working on building an intelligent agent to help a friend solve a problem that can be modeled using an MDP. In this environment, there are 3 possible states $S = \{s_1, s_2, s_3\}$ and at each state the agent always has 2 available actions $A = \{f, g\}$. Applying any action a from any state s has a probability $T(s, a, s')$ of moving the agent to one of the *other two* states but will never result in the agent staying at the original state. The rewards for this environment represent the cost/reward of an action, and thus are only dependent on the original state and action taken ($\forall s' \in S, R(s, a, s') = R(s, a)$), not where the agent ended up.

(a) (2 points) Write down the problem-specific Bellman update equations for each of the 3 states ($V(s) =?$) in this particular MDP. (Use the names of the specific states and actions.)

$$V(s_1) = \max_{a=\{f,g\}} \sum_{s'}^{s_2,s_3} T(s_1, a, s') \cdot [R(s_1, a) + \gamma V^*(s')].$$

$$V(s_2) = \max_{a=\{f,g\}} \sum_{s'}^{s_1,s_3} T(s_2, a, s') \cdot [R(s_2, a) + \gamma V^*(s')].$$

$$V(s_3) = \max_{a=\{f,g\}} \sum_{s'}^{s_1,s_2} T(s_3, a, s') \cdot [R(s_3, a) + \gamma V^*(s')].$$

(b) (8 points) One fateful day, while Jim was running a VI-based MDP solver on this problem, a mistake in specifying arguments caused the file that recorded the transition probability table $T(s, a, s')$ to be overwritten with the output solution. Now, Jim has the solution $V^*(s)$ and optimal policy $\pi^*(s)$ but has lost the transition probabilities for the problem.

s	a	$R(s, a)$	$V^*(s)$	$\pi^*(s)$
s_1	f	-3	-1.4	g
s_1	g	-4	-1.4	g
s_2	f	3	3	f
s_2	g	3	3	f
s_3	f	1.92	1.4	f
s_3	g	1.7	1.4	f

After looking at the command line history and noting that a discount of $\gamma = 1$ was specified, Jim muses that it may be possible to recover some parts of the transition probability table $T(s, a, s')$. Using the information above, fill in the values in table below that you

can recover, writing a "X" in the cells that you cannot produce a value for. Show your work by explaining how you recovered the values in the box under the table. For values that were derived in a similar way, it's sufficient to reference the previous explanation without repeating it again.

s	a	$T(s, a, s_1)$	$T(s, a, s_2)$	$T(s, a, s_3)$
s_1	f	0	X	X
s_1	g	0	0.75	0.25
s_2	f	0.5	0	0.5
s_2	g	X	0	X
s_3	f	0.8	0.2	0
s_3	g	X	X	0

First, I tried to substitute the values from the table to the equations and simplify the equations. Then based on the policy we know which move we should choose, or in other words, which results in larger expected utility. Then since we know the sum of the transition values for one move must be 1, so we can substitute one value into 1 minus the other. (ex. $T(s_1, f, s_2) = 1 - T(s_1, f, s_3)$). Solving for the value also gives us the other transition value based on this relationship. Lastly, we cannot obtain the transitional values for the not optimal actions $A(s_1)$, it's impossible to stay in the same state, so the probabilities are 0.

(c) (8 points) After some more thinking, Jim realized that while some values in the transition table cannot be ascertained with certainty, it is possible to instead figure out a reasonable *range* that the values would have been within. Using the same information as before, fill in places where a range can be derived for the transition probability (places where you marked an "X" on the table before) by noting the upper and lower bounds for valid values. (You may leave cells with fully recovered values from the previous part blank.) Show your derivation of these ranges in the box below the table. As before, if some ranges are derived in a similar way, you can just reference the previous explanation without having to repeat it.

s	a	$R(s,a)$	$V^*(s)$	$\pi^*(s)$
s_1	f	-3	-1.4	g
s_1	g	-4	-1.4	g
s_2	f	3	3	f
s_2	g	3	3	f
s_3	f	1.92	1.4	f
s_3	g	1.7	1.4	f

s	a	$T(s,a,s_1)$	$T(s,a,s_2)$	$T(s,a,s_3)$
s_1	f		$[0, 0.125)$	$(0.875, 1]$
s_1	g			
s_2	f			
s_2	g	$(0.5, 1]$		$[0, 0.5)$
s_3	f			
s_3	g	$(0.75, 1]$	$[0, 0.25)$	

We know that the sum of the expected utilities from this action that is not chosen must be lower than that of the optimal action. Thus we can set up an inequation using $V^*(s) > \sum$ utilities of action not chosen. Same as before, we can substitute one transition probability into (1 − probability of the other) and solve for that probability. When we get the result, if the probability is greater than the value, give it an upper limit of 1. If it's the reverse, give it a lower limit of 0 because the probability must lie between 0 & 1. Using this inequation, we can solve for the other probability.

(d) (2 points) If Jim had instead solved for Q-Values rather than V and π, what difference would it make to the ability to recover the transition probabilities $T(s,a,s')$? Explain why this is the case.

If we have the Q values, we can fully recover all transition probability because we have the values resulted from each action. Using the Q values, we can plug the values into our Bellman equations and solve for the unknown probabilities.