CS 188
Fall 2022

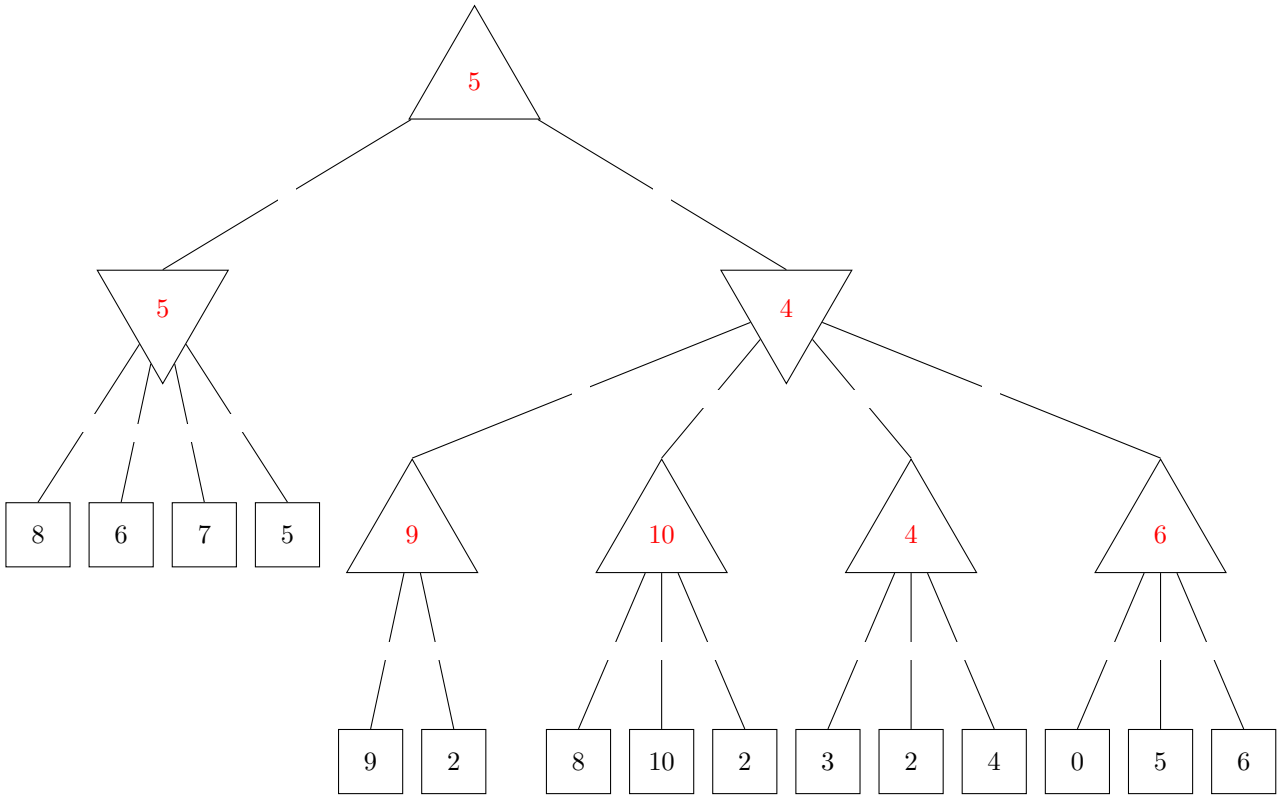
Regular Discussion 4 Solutions

Q1. Game Trees

The following problems are to test your knowledge of Game Trees.

(a) Minimax

The first part is based upon the following tree. Upward triangle nodes are maximizer nodes and downward are minimizers. (small squares on edges will be used to mark pruned nodes in part (ii))



(i) Complete the game tree shown above by filling in values on the maximizer and minimizer nodes.

(ii) Can any edges be pruned? Explain.

Edges that can be pruned: 10-2, 4-6

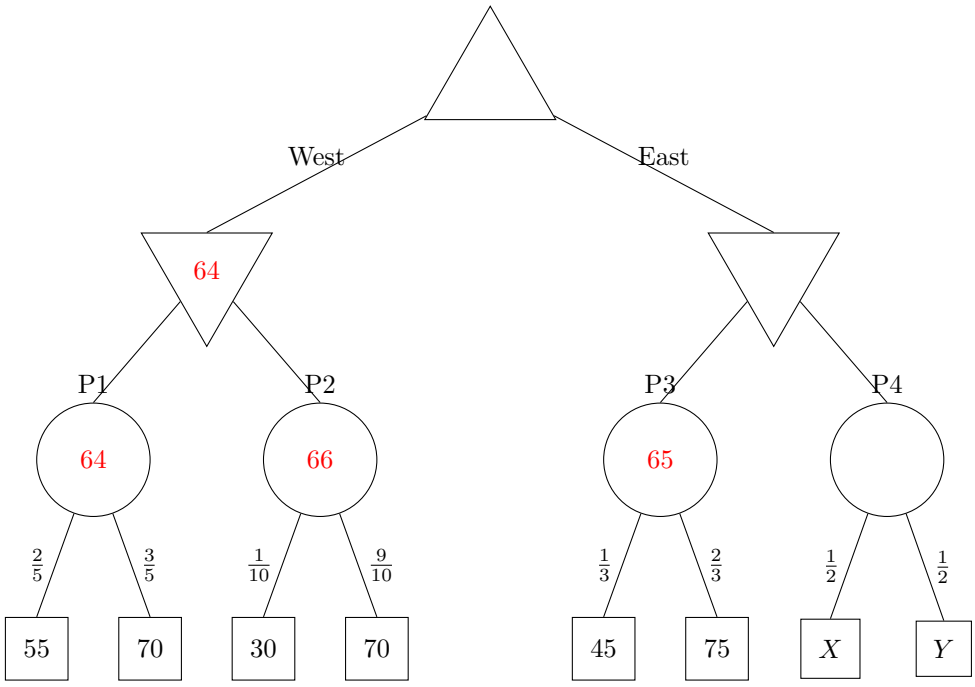
(b) Food Dimensions

The following questions are completely unrelated to the above parts.

Pacman is playing a tricky game. There are 4 portals to food dimensions. But, these portals are guarded by a ghost. Furthermore, neither Pacman nor the ghost know for sure how many pellets are behind each portal, though they know what options and probabilities there are for all but the last portal.

Pacman moves first, either moving West or East. After which, the ghost can block **1** of the portals available.

You have the following gametree. The maximizer node is Pacman. The minimizer nodes are ghosts and the portals are chance nodes with the probabilities indicated on the edges to the food. In the event of a tie, the left action is taken. Assume Pacman and the ghosts play optimally.



(i) Fill in values for the nodes that do not depend on X and Y.

(ii) What conditions must X and Y satisfy for Pacman to move East? What about to definitely reach the P4? Keep in mind that X and Y denote numbers of food pellets and must be **whole numbers**: $X, Y \in \{0, 1, 2, 3, \dots\}$.

To move East: $\boxed{X + Y > 128}$

To reach P4: $\boxed{X + Y = 129}$

The first thing to note is that, to pick A over B, $value(A) > value(B)$.
Also, the expected value of the parent node of X and Y is $\frac{X+Y}{2}$.
$\implies min(65, \frac{X+Y}{2}) > 64$
$\implies \frac{X+Y}{2} > 64$
So, $X + Y > 128 \implies value(A) > value(B)$

To ensure reaching X or Y, apart from the above, we also have $\frac{X+Y}{2} < 65$
$\implies 128 < X + Y < 130$
So, $X, Y \in \mathbb{N} \implies X + Y = 129$

Markov Decision Processes

A Markov Decision Process is defined by several properties:

- A set of states S
- A set of actions A.
- A start state.
- Possibly one or more terminal states.
- Possibly a **discount factor** γ.
- A **transition function** $T(s, a, s')$.
- A **reward function** $R(s, a, s')$.

The Bellman Equation

- $V^*(s)$ – the optimal value of s is the expected value of the utility an optimally-behaving agent that starts in s will receive, over the rest of the agent's lifetime.
- $Q^*(s, a)$ - the optimal value of (s, a) is the expected value of the utility an agent receives after starting in s, taking a, and acting optimally henceforth.

Using these two new quantities and the other MDP quantities discussed earlier, the Bellman equation is defined as follows:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

We can also define the equation for the optimal value of a q-state (more commonly known as an optimal **q-value**):

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

which allows us to reexpress the Bellman equation as

$$V^*(s) = \max_a Q^*(s, a).$$

Value Iteration

The **time-limited value** for a state s with a time-limit of k timesteps is denoted $V_k(s)$, and represents the maximum expected utility attainable from s given that the Markov decision process under consideration terminates in k timesteps. Equivalently, this is what a depth-k expectimax run on the search tree for a MDP returns.

Value iteration is a **dynamic programming algorithm** that uses an iteratively longer time limit to compute time-limited values until convergence (that is, until the V values are the same for each state as they were in the past iteration: $\forall s, V_{k+1}(s) = V_k(s)$). It operates as follows:

1. $\forall s \in S$, initialize $V_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence:

$$\forall s \in S,\ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

At iteration k of value iteration, we use the time-limited values for with limit k for each state to generate the time-limited values with limit $(k+1)$. In essence, we use computed solutions to subproblems (all the $V_k(s)$) to iteratively build up solutions to larger subproblems (all the $V_{k+1}(s)$); this is what makes value iteration a dynamic programming algorithm.

2 MDPs: Micro-Blackjack

In micro-blackjack, you repeatedly draw a card (with replacement) that is equally likely to be a 2, 3, or 4. You can either Draw or Stop if the total score of the cards you have drawn is less than 6. If your total score is 6 or higher, the game ends, and you receive a utility of 0. When you Stop, your utility is equal to your total score (up to 5), and the game ends. When you Draw, you receive no utility. There is no discount ($\gamma = 1$). Let's formulate this problem as an MDP with the following states: $0, 2, 3, 4, 5$ and a *Done* state, for when the game ends.

(a) What is the transition function and the reward function for this MDP? The transition function is

$$
\begin{aligned}
T(s, \mathit{Stop}, \mathit{Done}) &= 1 \\
T(0, \mathit{Draw}, s') &= 1/3 \text{ for } s' \in \{2, 3, 4\} \\
T(2, \mathit{Draw}, s') &= 1/3 \text{ for } s' \in \{4, 5, \mathit{Done}\} \\
T(3, \mathit{Draw}, s') &= \begin{cases} 1/3 \text{ if } s' = 5 \\ 2/3 \text{ if } s' = \mathit{Done} \end{cases} \\
T(4, \mathit{Draw}, \mathit{Done}) &= 1 \\
T(5, \mathit{Draw}, \mathit{Done}) &= 1 \\
T(s, a, s') &= 0 \text{ otherwise}
\end{aligned}
$$

The reward function is

$$
\begin{aligned}
R(s, \mathit{Stop}, \mathit{Done}) &= s, s \leq 5 \\
R(s, a, s') &= 0 \text{ otherwise}
\end{aligned}
$$

(b) Fill in the following table of value iteration values for the first 4 iterations.

States	0	2	3	4	5
V_0	0	0	0	0	0
V_1	0	2	3	4	5
V_2	3	3	3	4	5
V_3	10/3	3	3	4	5
V_4	10/3	3	3	4	5

(c) You should have noticed that value iteration converged above. What is the optimal policy for the MDP?

States	0	2	3	4	5
π^*	Draw	Draw	Stop	Stop	Stop