Homework 1

CSCI 360 Spring 2022

This assignment covers uninformed and informed search, game trees, CSPs, and reinforcement learning. Most questions of this assignment are graded, while some questions and parts will be marked as optional. Optional questions are extra practice for you to complete and won't count towards your homework grade.

Submission Instructions: You will receive an email and a Piazza reminder for enrolling in Gradescope, which is the platform we will be using for HW submissions and grading. To submit your solutions, first enroll in Gradescope and then submit your solution PDF. We recommend you type up your solutions using LaTeX but handwritten solutions are also fine. **Make sure that you correctly assign pages to questions when submitting to Gradescope, otherwise we may not be able to see and grade all of your answers.**

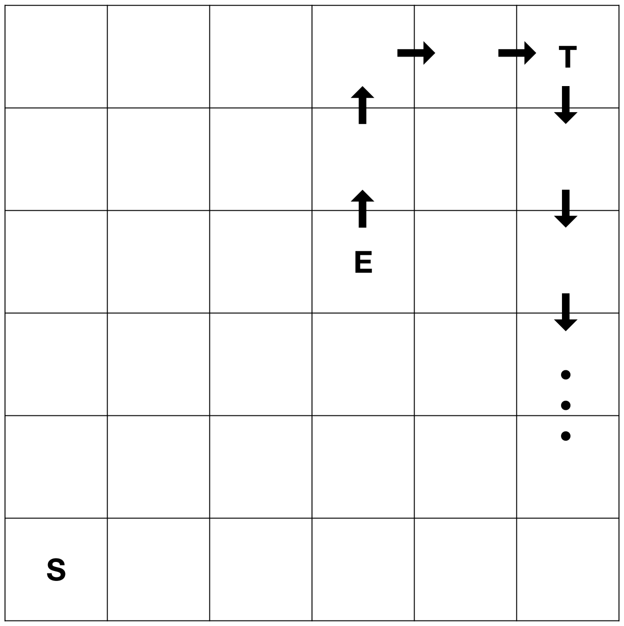
1 Uninformed Search and State Spaces [6 pts]

You are in charge of a new autonomous food delivery service on campus. Being the knowledgable CSCI 360 student that you are, you are going to use your newly acquired knowledge from the class to design state spaces, successor functions, and goal tests to create a solvable search problem to plan a path for your robot delivery vehicles.

There is one problem, however. The constant construction on campus is causing issues for your delivery robots, and you would like to avoid construction zones if possible.

In this problem, denote **S** as your robot's starting location, **E** as the construction vehicle's location, and **T** as Tommy Trojan statue's location. Your goal is to deliver food to some students at **T**ommy Trojan's location, i.e. to get from **S** to **T**. You know the starting location of the construction vehicle **E**.

The construction vehicle **E** puts 1 square under construction during each time step from its current location, going around the campus by cycling in the direction North, East, South, and West as shown in the figure (it's a deterministic path). If all squares adjacent to the vehicle are under construction, then it does not move. Every square on the grid in which **E** has visited is considered under construction, and the vehicle will not ever move into a square already under construction. For example, if the vehicle is traveling North, and at the current square its North, East, and South squares are already under construction or are the edges of the grid, then it will move West instead.



There are costs associated with each move that you make starting from **S**: the cost of entering a square under construction is 3 as the terrain is hard to traverse, and the cost of entering a square not under construction is 1. Assume your grid size is **N** x **N**, and that you can traverse North, East, South, or West, as long as doing so won't take you off of the grid. Finally, also assume a maximum of N^2 timesteps.

Answer each of the following questions with a quick justification of your answer.

(a) (1 pt) What is the **smallest** state space representation possible (in terms of number of states)? What is its size in number of states? What's the successor function (what are the valid actions and their associated costs)? What's the goal test?

State space representation:

We need both our current location and the current timestep of the environment. The location can be just an x-y coordinate, and timestep can be just a number. We don't need to represent the construction vehicle's entire previous path, because we know its starting location and the shape of the grid, and it takes a deterministic path from the starting location.

An alternate correct solution is to use the x-y coordinate of the current location and the x-y coordinate of the construction vehicle, as we can also reconstruct the path it took given just the position. This would result in the same state space size.

State space size:

N^4. There are N^2 grid positions, and the maximum number of timesteps before you reach the goal is bounded by N^2, so the total state space size is N^4.

Successor function:

The successor function returns actions which allow us to move over one square in a given direction as long as it does not go outside of our grid, and it returns costs associated with those actions that are 1 if the current square is not under construction and 3 otherwise.

Goal Test:

Are we at (N, N)?

(b) (1 pt) What is the cost of the optimal path in the specific instance given in the figure above using **UCS Tree Search** if you were starting at **S** and moving to **T**?

14. UCS will find the optimal path, which is to go North 2 steps, then East 4 steps, North 3 steps, and then East 1 step to the goal.

(c) (1 pt) What is the cost of the path in the specific instance given in the figure above using **BFS Graph Search** if you were starting at **S** and moving to **T**? Assume that ties for which states to put onto the fringe are broken in the order North, East, West, South. Hint: Perform this on a 2×2 or 3×3 grid to see a pattern.

16. BFS will always first expand the North square, adding its North, then East, then South, then West, neighboring squares to the fringe. Because it uses a queue as a fringe (first in first out), this means that BFS will reach the goal state by expanding North until the edge of the grid, and then expanding East until the goal state is retrieved from the queue. The South and West squares on this trajectory will always be skipped after being popped off the fringe as they will be visited already during the graph search.

(d) (0.5 pt) Would we want to use adversarial game trees for this situation? Why/why not?

No, as the construction vehicle is not an adversarial agent. It travels a deterministic path regardless of where we're going. We therefore just account for that in our state space.

(e) (0.5 pt) Now, assume that at each timestep, you have the power to freeze the construction vehicle for 1 timestep, by reporting to the administration that they don't have a valid Trojan Check. You can freeze the vehicle at the same time as you move on the grid. Does the smallest state space representation need to change for guaranteeing that we can find the optimal solution? If so, how would it change?

No, if we're trying to find the optimal solution, then we can report them at every timestep and not encode whether we reported them as part of our state.

Alternate correct solution because of possible wording confusion: Yes, we can just remove anything that keeps track of the location of the construction vehicle, since we will freeze it at every timestep.

(f) (Optional) Now, assume that there is no construction going on, and that you need to visit every spot on the grid at least once to deliver food to every spot on the entire campus. However, you have no idea where you are on campus, but you still need to guarantee that you have visited every single square. During the entire search, you will not know where you are. If you attempt to take a step out of bounds of the grid, then you will just stay in the same spot. Assume an unbounded number of timesteps for this task now, and assume that entering every square has the same cost. What is the **smallest** state space representation possible that guarantees we can still find the optimal solution? What is its size in number of states? What is the new goal test?

Below we list a couple of candidate solutions:

- **State Space Representation:** Since we don't know where we are, we need a boolean for every possible visited location from every possible start location. We would need to perform a search that keeps track of all possible start and end states and ensures that we have visited every square from every possible start state. Furthermore, for each of these start/end state configurations, we need to track which position we're currently in given that start state.

  State space size:
  $2^{N^4} \times (N^2)^{N^2}$. We need N^2 sets of N^2 booleans for a total of N^4 booleans. Furthermore, we need an extra (x, y) position tracker for each of the possible start states, resulting in N^2, position trackers which can each take on N^2 values.

  Goal Test:

  Check if all booleans are 1.

- **State Space Representation:** We can alternatively use a ternary representation, where the 0/1 values still represent which locations we have visited like above, but we also could use the value 2 to tell us the current position on the board as opposed to keeping track of the extra (x, y) positions for all possible start states.

  State space size: 3^{N^4}

  Goal Test: Check if all spots have a 1 or 2 in them.

- **State Space Representation:** We can simply record the action sequences and use that as the state representation.

  State space size: The length of the optimal action sequence is bounded by $4N^2$, the possible number of states is bounded by $4 + 4^2 + \cdots + 4^{4N^2} = \frac{4^{4N^2+1}-1}{3} = O(16^{N^2})$, which is smaller than the two solutions above.

  Goal Test: Use the action sequence to produce the N^2 sets of N^2 booleans in the first solution. Check if all booleans are 1.

(g) (1 pt) Now, assume the same setup as in the previous part, except you can only move in any direction (North, South, East, or West) a maximum of three times before needing to change directions. Assume that in the starting state, we have already moved North one time before starting. What is the **smallest** state space representation possible that guarantees we can still find the optimal solution? What is its size in number of states? What is the successor function (when will it let us move in which directions, and what will the states returned look like)? Use constant **K** to denote the state space size from the previous part.

State Space Representation:

We need the same booleans and position trackers as in the previous part, except now we also need a counter that keeps track of the number of times we have moved in the same direction, along with a value that stores what the previous direction was.

NOTE: If students attempted to give state space descriptions that accounted for (f) and they were incorrect but their state space size was correct, then we will give full credit.

State space size:

$K \times 4 \times 3$, the 4 coming from the 4 possible directions we previously traveled, the 3 coming from the number of times we have moved in that direction.

Successor Function:

The successor function returns actions which allow us to move over one square in a given direction as long as we have not exceeded 3 movements in that direction. Otherwise, it gives us states which we can move to in one of the 3 other directions, and those states will have the new position, direction we traveled to get there, and a 1 in the location that tracks the number of times we have moved in that direction.

(h) (1 pt) Finally, assume that there are M robots from your delivery company navigating around the campus. Assume the same setup as in part (f) except that you know the positions of all robots. But, you have limited cloud compute (it's expensive!), so each robot must take turns moving (e.g. the first robot moves while all others are frozen, then the next robot moves while all others are frozen, and so on). What is the new state space representation and size? The goal is still to have every location on campus visited by at least one robot.

State Space Representation:

We need the location of all M robots, a boolean for each position on the grid marking whether it's been visited, and a variable representing which robot's turn it is.

State Space Size:

$(N^2)^M \times 2^{N^2} \times M$, where $(N^2)^M$ represents the location of all M robots, 2^{N^2} represents booleans indicating which positions have been visited, and M for which robot's turn it is.

2 Informed Search and Heuristics [8 pts]

1. (2 pts) You are tasked with delivering packages in an M by N grid space, with each cell corresponding to a location. Suppose you start at a given arbitrary starting location, **S**. You need to deliver K packages. You have to pick up each package from their locations $P_1, P_2, ...P_K$ and deliver to their corresponding locations $D_1, D_2, ...D_K$ i.e, package picked at location P_i should be delivered to location D_i only. Each location P_i or D_i represents one cell location. The cost of moving from one cell to another is 1. Assume that you are allowed to move up, right, down, or left only. You can hold only 1 package at a time, i.e, your capacity is 1. When you enter a square with a package that has not yet been delivered, then you will automatically pick it up as long as you are not already carrying a package.

   (a) What is the size of the minimal state space? $(K+1)(MN)^{K+1}$ OR $(K+1)2^K MN$, as we failed to specify if $MN > 2$ or $2 > MN$.

   We need a variable to keep track of which package we are carrying or if we are empty-handed. It has $(k+1)$ possible values. Along with that, your starting location and K packages each can be any one of the MN cells available. Hence the total space size is $(K+1)(MN)^{K+1}$. Alternatively, you have your starting location and a boolean for whether each package has been delivered, which results in the alternate answer of $(K+1)MN2^K$

   (b) What is the maximum possible number of actions available at any given state? 4.

   Since your actions can be in any of the four directions only.

   (c) Let d(A, B) be the Manhattan distance between locations A and B. Which of the following heuristics given below is admissible from the starting location? Select all that apply.

   (i) $d(S, P_1) + d(S, P_2) + d(S, P_3) + \ldots + d(S, P_K)$
   (ii) $d(P_1, D_1) + d(P_2, D_2) + d(P_3, D_3) + ... + d(P_K, D_K)$
   (iii) $min\{d(P_1, D_1), d(P_2, D_2), d(P_3, D_3), ..., d(P_K, D_K)\}$
   (iv) $max\{d(P_1, D_1), d(P_2, D_2), d(P_3, D_3), ..., d(P_K, D_K)\}$
   (v) none of the above

   (ii), (iii), (iv)

   options ii, iii, iv will always be less than the total cost of optimal path

   (d) Suppose, you are now allowed to move diagonally as well. Which of the following heuristics given below is admissible from the starting location? Select all that apply.

   i. $d(S, P_1) + d(S, P_2) + d(S, P_3) + \ldots + d(S, P_K)$
   ii. $d(P_1, D_1) + d(P_2, D_2) + d(P_3, D_3) + ... + d(P_K, D_K)$
   iii. $min\{d(P_1, D_1), d(P_2, D_2), d(P_3, D_3), ..., d(P_K, D_K)\}$
   iv. $max\{d(P_1, D_1), d(P_2, D_2), d(P_3, D_3), ..., d(P_K, D_K)\}$
   v. none of the above

   (v)

   Since you are now allowed to move diagonally, the optimal path will include euclidean distance between the locations in the path. $d(A, B)$ is the Manhattan distance which is $\geq$ Euclidean distance between A and B. Hence none of the above options are admissible.

   (e) Suggest an admissible heuristic when (d) is allowed. Replace the above Manhattan distance calculation with euclidean distance

2. (2 pts) Consider the following problem with a given set of elements A, B, C, D, E. Given a start state say: [B, C, A, D, E], reach the goal state: [A, B, C, D, E] using the following operation: Pick a pivot point and reverse the order from the starting point till the pivot point. For example, if we pick pivot point 2 (indexed starting from 0) in the above start state, performing the operation will change the state from [B, C, A, D, E] to [A, C, B, D, E].

   (a) Given any arbitrary start state, what is the size of minimal state space? 5!

   A state can be any one of the 5! permutations available for A, B, C, D, E

   (b) What is the maximum possible number of actions available at any given state? Consider only those actions that result in a new state different from current state. 4

   Operation performed at index 1, 2, 3, 4 would result in a new states, hence 4 actions are available for any given state

   (c) If the cost of performing the operation is equal to 1, is the following heuristic admissible? h(s): number of elements not in their desired positions No

   For state {D, C, B, A, E} to become the goal state {A, B, C, D, E}, it requires only one operation - pivot at index 3. Hence it costs only one. However the heuristics of this action would be 4. Therefore, h(s) is not admissible

   (d) If the cost of performing the operation is equal to the number of elements reversed, is the above heuristic admissible? Yes

3. (2 pts) Consider the following graph. Perform A* search with start state as S and goal state as G. In case of ties, choose the nodes in lexicographical order.

   (a) Tree search

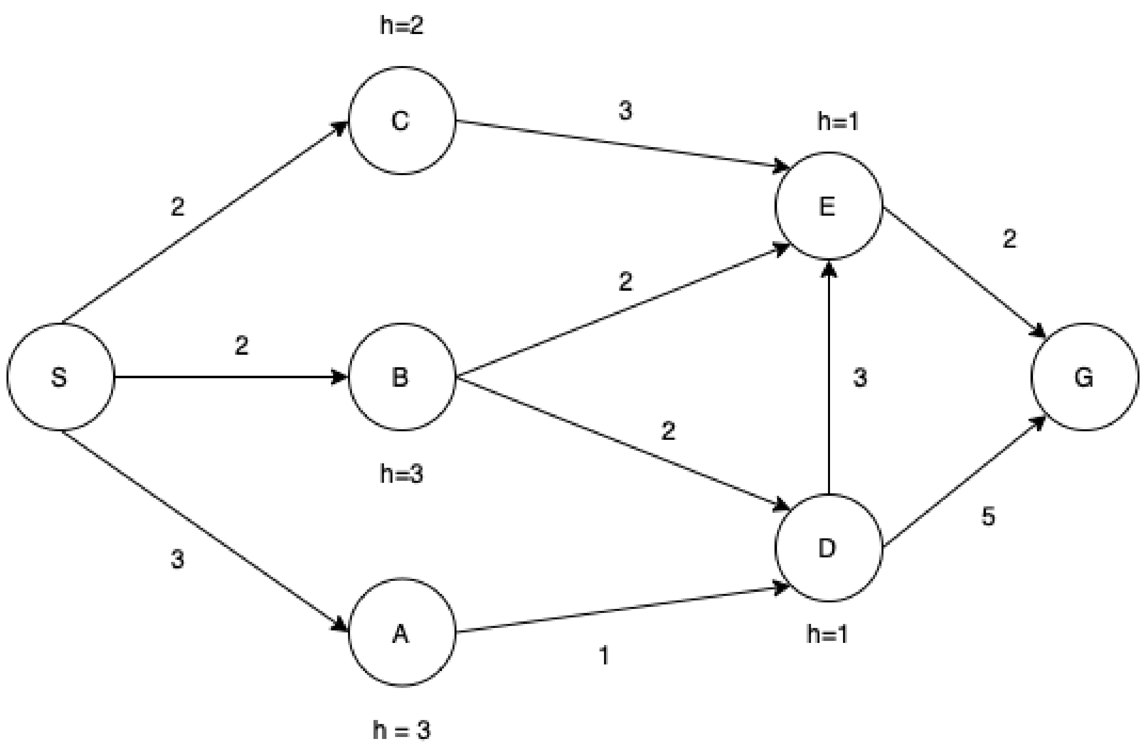
      (1) List the nodes in the order they are expanded from the queue. Each node should be represented with its full path. S, (S)C, (S)B, (S->B)D, (S->B)E, (S)A, (S->A)D, (S->C)E, (S->B->E)G

      (2) Give the final solution path obtained by the algorithm and its cost. S->B->E->G, cost: 6

   (b) Graph Search

      (1) List the nodes in the order they are expanded from the queue. Each node should be represented with its full path. S, (S)C, (S)B, (S->B)D, (S->B)E, (S)A, (S->B->E)G

      (2) Give the final solution path obtained by the algorithm and its cost. S->B->E->G, cost: 6



(c) Is the heuristic admissible? If not, what could be changed to make it admissible?
heuristic is admissible

(d) Is the heuristic consistent? If not, what could be changed to make it consistent?
heuristic is inconsistent. h(A) > (cost of A to D) + h(D).

- h(A) can be reduced to 2 or less.
- or h(D) can be increased to any value between 2 to 5.
- or cost of A to D can be increased to 2 or more

4. (2 pts) Suppose we are conducting A* tree search. We already have an **admissible** heuristic h(x), but now we want to test the algorithm with a new heuristic $h'(x)$. Let $h^*(x)$ be the optimal cost of path to reach the goal state from state x. Let C be the cost of path obtained using the A* search with heuristic $h'(x)$. For each of the relation between $h'(x)$ and h(x) listed below, select the option that is correct for the relation between C and $h^*(x)$. You an assume all heuristic values are at least 0.

(a) $h'(x) = h^*(x) - h(x)$

(i) $C > h^*(S)$ (ii) $C = h^*(S)$ (iii) $C \geq h^*(S)$

(b) $h'(x) = h^*(x)$

(i) $C > h^*(S)$ (ii) $C = h^*(S)$ (iii) $C \geq h^*(S)$

(c) $h'(x) = h(x)^{0.99}$

(i) $C > h^*(S)$ (ii) $C = h^*(S)$ (iii) $C \geq h^*(S)$

(d) $h'(x) = h(x) + h^*(x)$

(i) $C > h^*(S)$ (ii) $C = h^*(S)$ (iii) $C \geq h^*(S)$

(e) $h'(x) = \sqrt{h(x) * h^*(x)}$

(i) $C > h^*(S)$ (ii) $C = h^*(S)$ (iii) $C \geq h^*(S)$

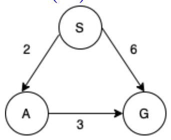
(a) (ii) Since $h'(x)$ is less than $h^*(x)$, i.e, $h'(x)$ is admissible, C would be optimal cost

(b) (ii)

(c) (iii)

(d) (iii). Since $h'(x)$ is greater than $h^*(x)$, C may or may not be optimal cost. When there exists only one path from start to goal state, any heuristic will lead to optimal path. hence option (i) is ruled out.

For option (ii) consider the counter example with the graph below. S is the start state and G is the goal state $h^*(A) = 3$, $h(A) = 1$, $h'(A) = 4$, $h^*(S) = 5$. A* search will find the optimal



$$\begin{tikzcd} & S \arrow[dl, "2"'] \arrow[dr, "6"] & \\ A \arrow[rr, "3"'] & & G \end{tikzcd}$$

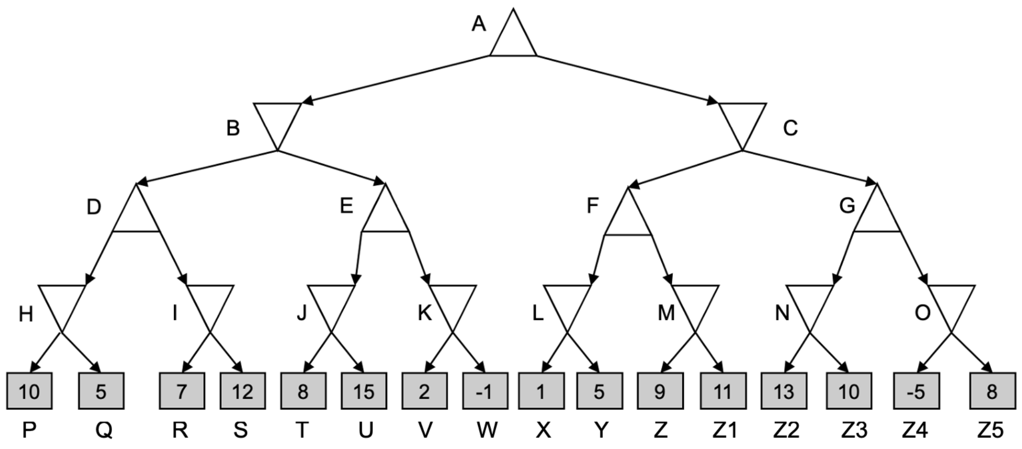
path for the above example, i.e, here $C = h^*(x)$

(e) (ii). Since $h'(x)$ is be less than $h^*(x)$, C would be optimal cost

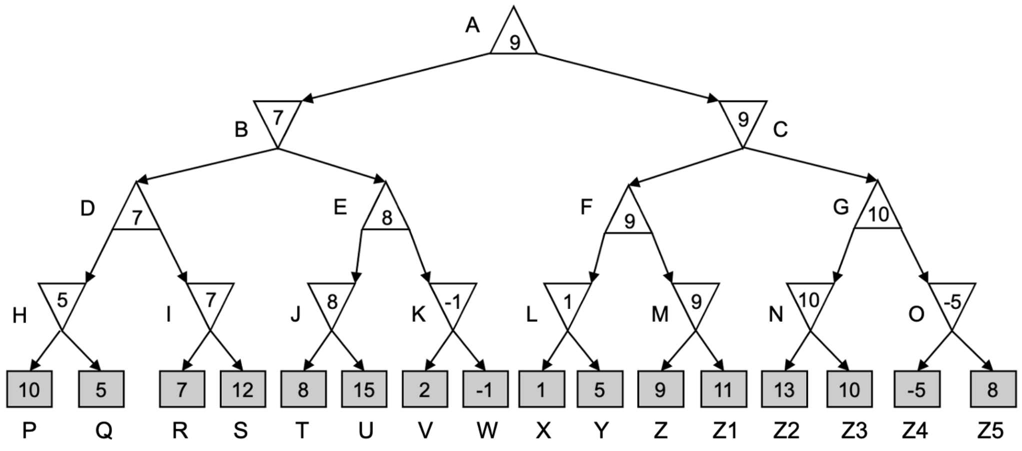
3 Game Trees [7 pts]

1. (3 pts) Consider a zero-sum game in which there are n coins in a line (assume that n is even here). Two people take turns to take a coin from one of the ends of the line until there are no more coins left. The person with the larger amount of money wins, and there can be negative and positively valued coins with differing values.

   (a) Consider the game tree for the above game as shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Assuming both players act optimally, write the minimax value of each node (A through O).



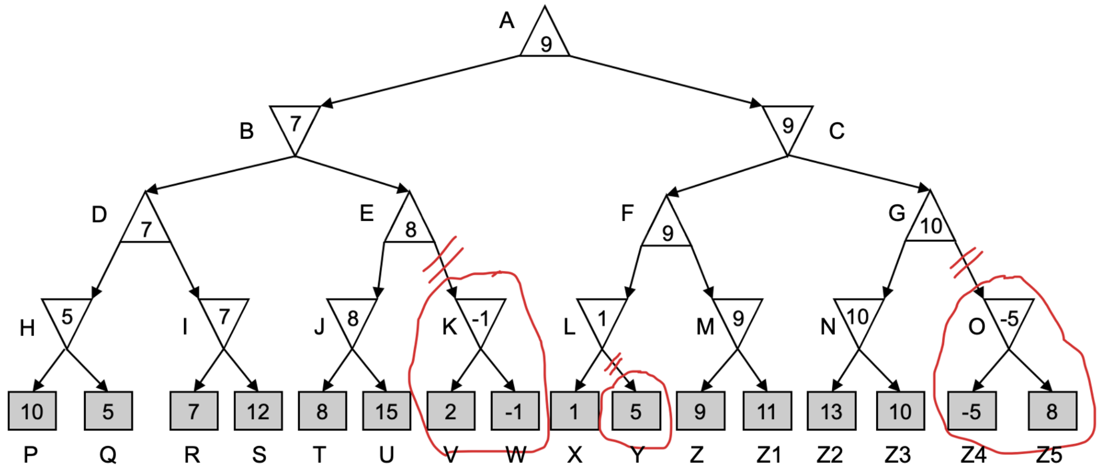
Each downward pointing triangle is considered as the minimizing function, whereas each upward pointing triangle is considered as the maximizer funtion.
So if we start from the leftmost nodes P and Q and apply the minimizer function here, we get 5. Following this we can fill up all the nodes in the tree as shown below.



(b) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. Assume the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.

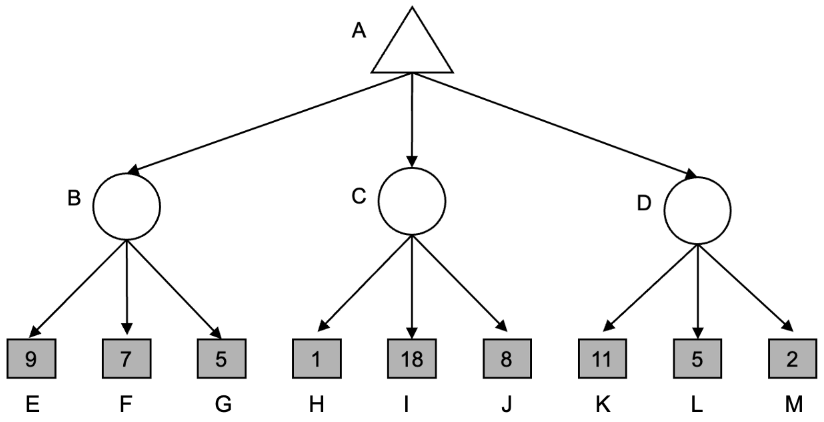
In this subproblem, we perform alpha beta pruning on the above tree. We need to check the primary conditions of alpha>=beta to perform pruning. We intialize alpha to -infinity and beta to +infinity and follow the alpha-beta pruning algorithm to compute the pruned nodes.

Lets start with the leftmost nodes P and Q. We check if the value of node P(10) is less than infinity and update the lower value of beta, since we are performing the minimize function here. We then check for node Q and update the beta value subsequently. We keep performing this process till we get a condition of alpha greater than or equal to beta which will be when we consider nodes T and U. Here the value of alpha and beta will be 7. To visualize better let's consider nodes T and U, here the max for node E is 8, and the maximum of the other branch is already 7 (Node D). Now if we consider this state, the next function will be minimize operation between a value of 7 and anything atleast as low as 8. Therefore, the values of the next tree (Nodes K, V and W here) are not considered (pruned) since whatever the final value, the least value that will be considered would be 8, which is more than 7 and hence the minimize function will always chose 7 (Value of Node D). We follow the same process for all the other nodes and prune the nodes when the condition of alpha >= beta is met.

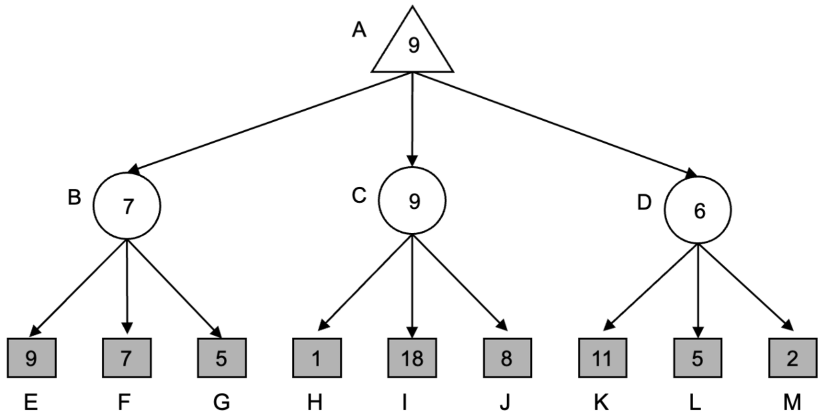


2. (4 pts) Let us consider a similar 3-level zero-sum game tree, except that now, instead of a minimizing player, we have a chance node that will select one of the three values uniformly at random.

(a) Fill in the Expectimax value of each node in the tree. The game tree is shown below for your convenience. Please show the calculation for each node(A through D).



The expectimax values for each chance node can be calculated as a weighted average of all the child nodes. The calculation is shown below.
B=(9+7+5)/3=7
C=(1+18+8)/3=9
D=(11+5+2)/3=6
A = Max(7,9,6) = 9
Each child will have an equal probability of 1/3



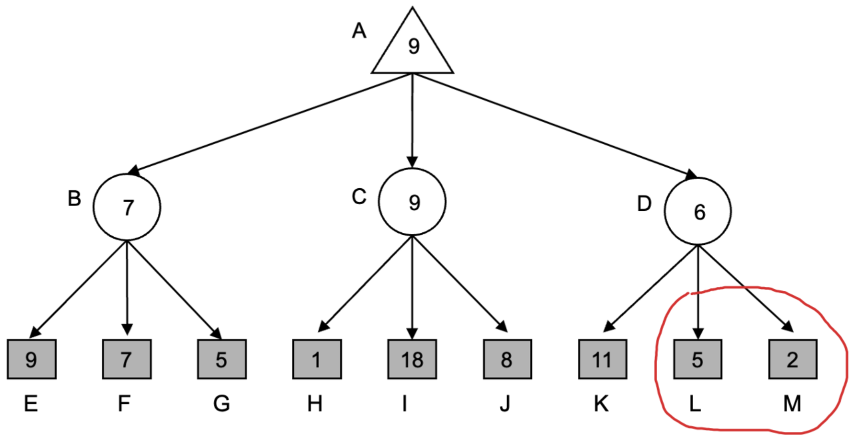
(b) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not.

No nodes can be pruned. There will always be the possibility that an as-yet-unvisited leaf of the current parent chance node will have a very high value, which increases the overall average value for that chance node. For example, when we see that node H has a value of 1, which is much less than the value of the left chance node B, 7, at this point we cannot make any assumptions about how the value of the middle chance node C will ultimately be more or less in value than the node B. As it turns out, the node I has a value of 18, which brings the expected value of the middle chance node to 9, which is greater than the value of the node B. In the case where there is an upper bound to the value of a leaf node, there is a possibility of pruning:

suppose that an upper bound of +5 applies only to the children of the rightmost chance node. In this case, after seeing that node K has a value of 11 and node L has a value of 5, the best possible value that the rightmost chance node D can take on is (11+5+5)/3 = 7, which is less than 9, the value of the middle chance node. Therefore, it is possible to prune leaf node M in this case.

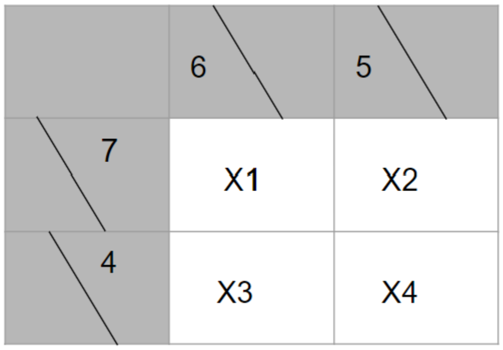
(c) Which nodes can be pruned from the game tree above through alpha-beta pruning if there is a bound of +6 on the nodes L and M? Justify your answer.

The leaf nodes L and M would be pruned, since they have an upper bound of +6 which means that the maximum value of either of those nodes will not be greater than 6. Therefore, we can prune Nodes L and M because when node K is explored, we know than the combined max value of two other nodes is 12. If we consider this value, the value of D will at max be (11+6+6)/3 which is less than the value of chance node C which is 9. Hence, at this point we can prune nodes L and M since whatever their value, it will not have any impact on the value of node A.



4 Constraint Satisfaction Problems [6 pts]

Kakuro is a logic puzzle that is often referred to as a mathematical transformation of a crossword puzzle. Just like a crossword it consists of a grid of squares as below:



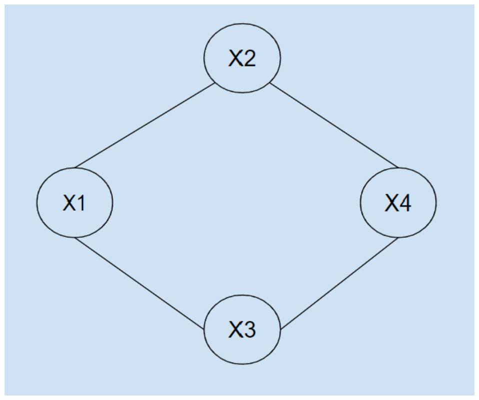
A set of connected horizontal or vertical blank squares is called a block. The objective is to fill all empty squares using numbers 1 to 9 so the sum of each horizontal block equals the clue on its left, and the sum of each vertical block equals the clue on its top. In addition, no number may be used in the same block more than once.

For instance, in the above puzzle, X1 and X2 form a horizontal block and their sum should equal 7. Also, you cannot use a digit more than once in a block. For example, you cannot use digit 3 twice in the vertical block above leading to 6, to get sum 6.

(a) (1 pt) Formulate the puzzle above as a CSP with variables, domains, and constraints.

Consider a variable for each blank square: X1, X2, X3, X4.
Domain for all variables: {1, 2, ...9}
Constraints:
X1 + X2 = 7 , X1 != X2
X1 + X3 = 6 , X1 != X3
X3 + X4 = 4 , X3 != X4
X2 + X4 = 5 , X2 != X4

(b) (1 pt) Draw the constraint graph representing the above CSP.



(c) (2 pts) Run 2 steps of arc consistency (where one step corresponds to popping one arc off of the arc consistency queue). Pop arcs off of the queue in numerical order of the tail, then head, variables (e.g. $X1 \rightarrow X2$ is popped before $X1 \rightarrow X3$ which would be popped off before $X2 \rightarrow X1$).

1. What arcs are currently on the queue?
   $X2 \rightarrow X1$
   $X2 \rightarrow X4$
   $X3 \rightarrow X1$
   $X3 \rightarrow X4$
   $X4 \rightarrow X2$
   $X4 \rightarrow X3$
   The arcs that have been removed are $X1 \rightarrow X2$ and $X1 \rightarrow X3$, no new arcs have been added to the agenda.
2. What are the current domains of all considered variables?
   X1 = 1, 2, 4, 5 (Students must remove 3 from the domain of X1 when popping $X1 \rightarrow X3$ to satisfy inequality constraint X1 != X3)
   X2 = 1, 2, 3, 4, 5, 6, 7, 8, 9
   X3 = 1, 2, 3, 4, 5, 6, 7, 8, 9
   X4 = 1, 2, 3, 4, 5, 6, 7, 8, 9

(d) (1 pt) Show the domains of all considered variables after completing arc-consistency.
X1 = 5
X2 = 2
X3 = 1
X4 = 3

(e) (1 pt) Let's say the current domains after some steps of arc consistency are as follows: $X1 = \{5, 1, 2\}$, $X2 = \{2, 3, 4\}$, $X3 = \{1, 3\}$, $X4 = \{3, 4, 5\}$. If we were to run search from this point, name what variable would be chosen for assignment with the Minimum Remaining Values ordering strategy:

X3

5 Reinforcement Learning and MDPs [7 pts]

In pseudo-blackjack, you will keep drawing cards from a deck (with replacement). In this simplified version, the only cards in the deck are a 2, 3, or 4. At each timestep, you can choose to either "Hit" (request a card from the dealer) or "Stand" (stop playing for the current round). If the sum of the values of the cards you hold are at least 6, then the round ends and you will receive a reward of 0. When you Stand, your reward will be equal to the sum of your card values and the round comes to an end. When you Hit, you receive no additional reward. Assume no discount, i.e. $\gamma = 1$. Whenever the round comes to an end, assume you enter a special "Done" state from which you cannot take any actions.

(a) (0.5 pt) We will formulate this as a Markov Decision Process (MDP). Enumerate all states in the state space and all possible actions of this MDP.

The states are $\{0, 2, 3, 4, 5, Done\}$

The actions are Hit or Stand.

(b) (1.5 pts) Define the transition function ($T(s, a, s')$) and reward function ($R(s, a, s')$) for this MDP.

$T(s, Stand, Done) =$		
$T(0, Hit, s') =$		if $s' \in \{2, 3, 4\}$
$T(2, Hit, s') =$		if $s' \in \{4, 5, Done\}$
$T(3, Hit, s') =$		if $s' = 5$
$T(3, Hit, s') =$		if $s' = Done$
$T(4, Hit, s') =$		
$T(5, Hit, s') =$		
$T(s, a, s') =$		otherwise
$R(s, Stand, Done) =$		if $s \leq 5$
$R(s, a, s') =$		otherwise

$T(s, Stand, Done) =$	1	
$T(0, Hit, s') =$	1/3	if $s' \in \{2, 3, 4\}$
$T(2, Hit, s') =$	1/3	if $s' \in \{4, 5, Done\}$
$T(3, Hit, s') =$	1/3	if $s' = 5$
$T(3, Hit, s') =$	2/3	if $s' = Done$
$T(4, Hit, s') =$	1	
$T(5, Hit, s') =$	1	
$T(s, a, s') =$	0	otherwise
$R(s, Stand, Done) =$	s	if $s \leq 5$
$R(s, a, s') =$	0	otherwise

(c) (2 pts) Now we will perform value iteration for a fixed number of iterations. Fill in the following table for the values of each state ($V(s)$) at each iteration.

States	0	2	3	4	5
$V_0(s)$					
$V_1(s)$					
$V_2(s)$					
$V_3(s)$					
$V_4(s)$					

States	0	2	3	4	5
$V_0(s)$	0	0	0	0	0
$V_1(s)$	0	2	3	4	5
$V_2(s)$	3	3	3	4	5
$V_3(s)$	10/3	3	3	4	5
$V_4(s)$	10/3	3	3	4	5

Explanation: By default, we set all values to 0 initially, so $V_0(s) = 0\ \forall s$

For $V_1(s)$, all of the value iteration equations for all the states in the table look like this: $V_1(s) = \max\{1 * (R(s, Stand, Done) + V_0(Done)), \sum_{s'} T(s, a, s') * (R(s, Hit, s') + V_0(s'))\}$ (note that $\gamma = 1$ so it's excluded from the equation here). Note that we don't get any reward for hitting, and all $V_0(s')$ are 0, so the best action will be to Stand from all states to obtain some reward, except state 0. In state 0, there's a tie between the two actions so the best action to take there is arbitrary, as standing at state 0 gets us 0 for the left part of the max equation. Therefore, $V_1(s) = s\ \forall s \in \{2, 3, 4, 5\}$ and $V_1(0) = 0$. See the next page for further details.

For $V_2(s)$, we will go state by state.

$$\begin{aligned}
V_2(5) &= \max\left\{1 * (R(5, Stand, Done) + V_1(Done)), 1 * (R(5, Hit, Done) + V_1(Done))\right\} \\
&= \max\{5+0, 0+0\} \\
&= 5 \\
V_2(4) &= \max\left\{1 * (R(4, Stand, Done) + V_1(Done)), 1 * (R(4, Hit, Done) + V_1(Done))\right\} \\
&= \max\{4+0, 0+0\} \\
&= 4 \\
V_2(3) &= \max\{1 * (R(3, Stand, Done) + V_1(Done)), 1/3 * (R(3, Hit, 5) + V_1(5)) \\
&+ 2/3 * (R(3, Hit, Done) + V_1(Done)) \\
&= \max\{3, 1/3 * 5 + 2/3 * 0\} \\
&= 3 \\
V_2(2) &= \max\{1 * (R(2, Stand, Done) + V_1(Done)), 1/3 * (R(2, Hit, 5) + V_1(5)) \\
&+ 1/3 * (R(2, Hit, 4) + V_1(4)) + 1/3 * (R(2, Hit, Done) + V_1(Done)) \\
&= \max\{2, 1/3 * 5 + 1/3 * 4 + 1/3 * 0\} \\
&= \max\{2, 3\} \\
&= 3 \\
V_2(0) &= \max\{1 * (R(0, Stand, Done) + V_1(Done)), 1/3 * (R(0, Hit, 4) + V_1(4)) \\
&+ 1/3 * (R(0, Hit, 3) + V_1(3)) + 1/3 * (R(0, Hit, 2) + V_1(2)) \\
&= \max\{0, 1/3 * 4 + 1/3 * 3 + 1/3 * 2\} \\
&= 3
\end{aligned}$$

For $V_3(s)$ we can use the same formulas as from $V_2(s)$ except plug in the $V_2(s)$ instead of $V_1(s)$ in the equations.

$$\begin{aligned}
V_3(5) &= \max\{5, 0\} \\
&= 5 \\
V_3(4) &= \max\{4, 0\} \\
&= 4 \\
V_3(3) &= \max\{3, 1/3 * 5 + 2/3 * 0\} \\
&= 3 \\
V_3(2) &= \max\{2, 1/3 * 5 + 1/3 * 4\} \\
&= 3 \\
V_3(0) &= \max\{0, 1/3 * 4 + 1/3 * 3 + 1/3 * 3\} \\
&= 10/3
\end{aligned}$$

Note that at this iteration $V_3(5), V_3(4), V_3(3), V_2(2)$'s values didn't change from before. Furthermore, since these states cannot transition into state 0, the only state whos value changed, value iteration has converged for these states. So if we recalculate for V_4, the values will not change for these states either. We can just recalculate $V_4(0) = \max\{0, 1/3 * 4 + 1/3 * 3 + 1/3 * 3\} = 10/3$.

(d) (1 pt) Extract the optimal policy for this MDP (list the actions the policy takes at each state). The optimal policy is given by: $\pi^*(s) = \text{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + V(s')]$

States	0	2	3	4	5
$\pi^*(s)$					

States	0	2	3	4	5
$\pi^*(s)$	Hit	Hit	Stand	Stand	Stand

(e) (2 pts) Starting with the given policy below, perform one iteration of policy iteration (ignore values from previous steps). Hint, you will get values of $V^{\pi_0}(s)$ that depend on other $V^{\pi_0}(s')$ but the system of equations can be solved.

States	0	2	3	4	5
$\pi_0(s)$	Hit	Stand	Hit	Stand	Hit
$V^{\pi_0}(s)$					
$\pi_1(s)$					

States	0	2	3	4	5
$\pi_0(s)$	Hit	Stand	Hit	Stand	Hit
$V^{\pi_0}(s)$	2	2	0	4	0
$\pi_1(s)$	Hit	Stand	Stand	Stand	Stand

$$\begin{aligned}
V^{\pi_0}(5) &= T(5, Hit, Done) * [R(5, Hit, Done) + \gamma * V^{\pi_0}(Done)] \\
&= 1 * [0 + V^{\pi_0}(Done)] \\
&= 0 \\
V^{\pi_0}(4) &= T(4, Stand, Done) * [R(4, Stand, Done) + \gamma * V^{\pi_0}(Done)] \\
&= 1 * 4 \\
&= 4 \\
V^{\pi_0}(3) &= T(3, Hit, 5) * [R(3, Hit, 5) + \gamma * V^{\pi_0}(5)] \\
&+ T(3, Hit, Done) * [R(3, Hit, Done) + \gamma * V^{\pi_0}(Done)] \\
&= 1/3 * 0 + 2/3 * 0 \\
&= 0 \\
V^{\pi_0}(2) &= T(2, Stand, Done) * [R(2, Stand, Done) + \gamma * V^{\pi_0}(Done)] \\
&= 2 \\
V^{\pi_0}(0) &= T(0, Hit, 2) * [R(0, Hit, 2) + \gamma * V^{\pi_0}(2)] \\
&+ T(0, Hit, 3) * [R(0, Hit, 3) + \gamma * V^{\pi_0}(3)] \\
&+ T(0, Hit, 4) * [R(0, Hit, 4) + \gamma * V^{\pi_0}(4)] \\
&= 1/3 * 2 + 1/3 * 0 + 1/3 * 4 \\
&= 2
\end{aligned}$$

$$\begin{aligned}
\pi_1(5) &= \text{argmax}\{T(5, Hit, Done) * [R(5, Hit, Done) + \gamma * V^{\pi_0}(Done)]\,,\\
&\quad T(5, Stand, Done) * [R(5, Stand, Done) + \gamma * V^{\pi_0}(Done)]\}\\
&= \text{argmax}\{0, 5\}\\
&= Stand\\
\pi_1(4) &= \text{argmax}\{T(4, Hit, Done) * [R(4, Hit, Done) + \gamma * V^{\pi_1}(Done)]\,,\\
&\quad T(4, Stand, Done) * [R(4, Stand, Done) + \gamma * V^{\pi_0}(Done)]\}\\
&= \text{argmax}\{0, 4\}\\
&= Stand\\
\pi_1(3) &= \text{argmax}\{T(3, Hit, Done) * [R(3, Hit, Done) + \gamma * V^{\pi_0}(Done)]\\
&\quad + T(3, Hit, 5) * [R(3, Hit, 5) + \gamma * V^{\pi_0}(5)]\,,\\
&\quad T(3, Stand, Done) * [R(3, Stand, Done) + \gamma * V^{\pi_0}(Done)]\}\\
&= \text{argmax}\{2/3 * 0 + 1/3 * 0, 3\}\\
&= Stand\\
\pi_1(2) &= \text{argmax}\{T(2, Hit, Done) * [R(2, Hit, Done) + \gamma * V^{\pi_0}(Done)]\\
&\quad + T(2, Hit, 4) * [R(2, Hit, 4) + \gamma * V^{\pi_0}(4)]\\
&\quad + T(2, Hit, 5) * [R(2, Hit, 5) + \gamma * V^{\pi_0}(5)]\,,\\
&\quad T(2, Stand, Done) * [R(2, Stand, Done) + \gamma * V^{\pi_0}(Done)]\}\\
&= \text{argmax}\{1/3 * 0 + 1/3 * 4 + 1/3 * 0, 2\}\\
&= Stand\\
\pi_1(0) &= \text{argmax}\{T(0, Hit, 2) * [R(0, Hit, 2) + \gamma * V^{\pi_0}(2)]\\
&\quad + T(0, Hit, 3) * [R(0, Hit, 3) + \gamma * V^{\pi_0}(3)]\\
&\quad + T(0, Hit, 4) * [R(0, Hit, 4) + \gamma * V^{\pi_0}(4)]\,,\\
&\quad T(0, Stand, Done) * [R(0, Stand, Done) + \gamma * V^{\pi_0}(Done)]\}\\
&= \text{argmax}\{1/3 * 2 + 1/3 * 0 + 1/3 * 4, 0\}\\
&= Hit
\end{aligned}$$