CSE 415 Autumn 2022 **Assignment 4: Solutions**

PLEASE DO NOT SHARE OUTSIDE OF CSE 415 IN AUTUMN 2022

Last name:_____ First name:_____

Due Wednesday night May 5 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.

This is an individual-work assignment. The staff will not tolerate collaboration, even if some cases were overlooked in the past.

Prepare your answers in a neat, easy-to-read PDF. Our grading rubric will be set up such that when a question is not easily readable or not correctly tagged or with pages repeated or out of order, then points will be be deducted. However, if all answers are clearly presented, in proper order, and tagged correctly when submitted to Gradescope, we will award a 5-point bonus.

If you choose to choose to typeset your answers in Latex using the template file for this document, please put your answers in blue while leaving the original text black.

---

1 Blind Search with the Towers of Hanoi

The 2-disk version of the Towers of Hanoi is a trivial puzzle for humans and machines alike. However, it's a nice and simple context for comparing different algorithms.

Several aspects of this problem will come up again in Assignment 5, and so this problem will not only help you get more familiar with certain details of the search algorithms, but it will provide some insight into the the Towers of Hanoi problem space.

Let us assume that the problem is formulated with the following state representation and operators.

Initial state:

```
Left: 1,2
Middle:
Right:
```

Goal state:

```
Left:
Middle:
Right: 1,2
```

Operator ϕ_0: "Move a disk from Left to Middle."

Operator ϕ_1: "Move a disk from Left to Right."

Operator ϕ_2: "Move a disk from Middle to Left."

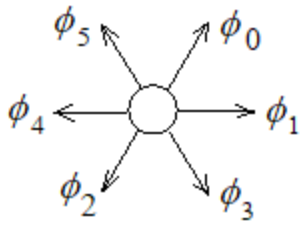
Operator ϕ_3: "Move a disk from Middle to Right."

Operator ϕ_4: "Move a disk from Right to Left."

Operator ϕ_5: "Move a disk from Right to Middle."

Hand simulate DFS (Depth-First Search, BFS (Breadth-First Search) and IDDFS (Iterative-Deepening Depth-First Search) on this problem, in order to determine the state visitation orderings and compare them.

The problem-space graph for this formulation can be laid out as in the diagrams for sub-questions a-c below. Note that the operators always take a specific direction, as shown in this diagram:



Use the copies of the problem space graph below to show the progress of each search algorithm. When doing IDDFS, you'll use a separate copy of the graph for each outer iteration.

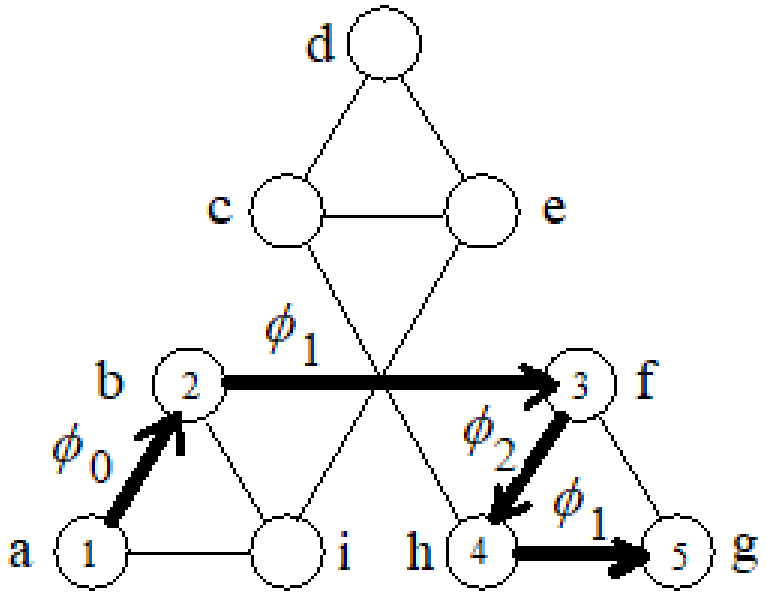
For each algorithm stop when the goal node (g) is selected as the current state.

Number the nodes as they are visited (i.e., as they become the current state). The initial state should get numbered 1 when each algorithm starts. However, during IDDFS, it should get multiple numbers since it will be visited multiple times.

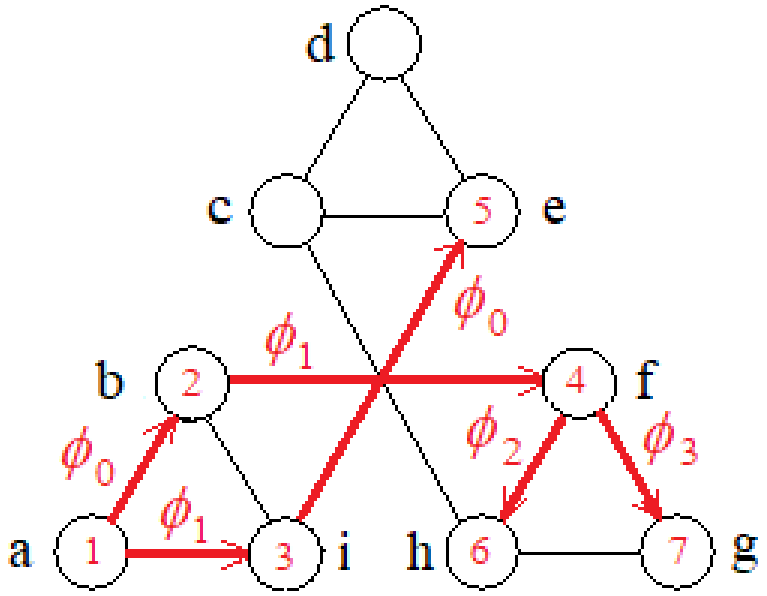
In order to get the correct numberings, it is very important to follow the pseudocode in the DFS, and BFS algorithms, and to generate the successors by using the operators in their given order: $\phi_0, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5$.

Each time a new node is reached in the search (i.e., put onto the OPEN list), draw an arrow to it from its parent. Note that this does not apply to the initial state, since it has no parent. When doing this for IDDFS, you'll retain (i.e., copy from one diagram to the next) these arrows from one outer iteration to the next. At this end of IDDFS, you should have an optimal path from the initial state to the goal state, which can be recovered by backtracing from the goal.

(a) (0 points) Hand-simulate DFS and put the node visitation order on the graph. Note that the answers to this part are done for you as an example.

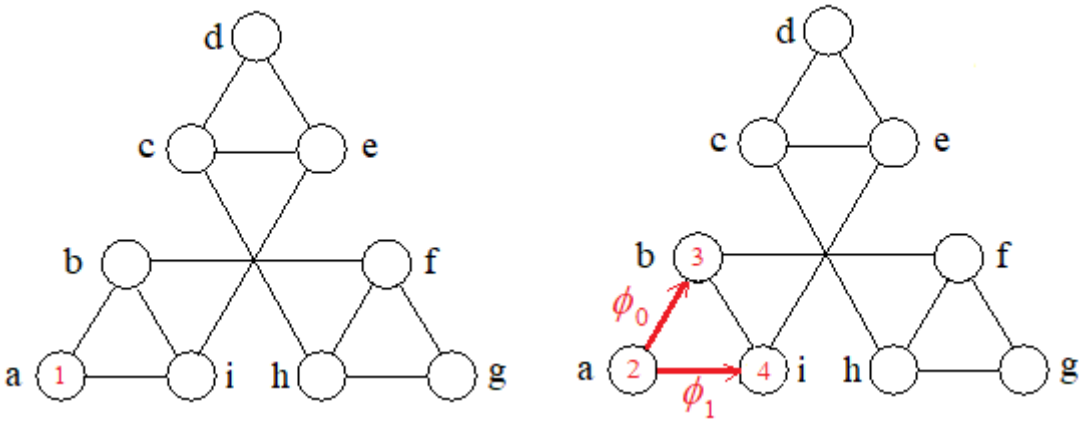


(b) (8 points) Hand-simulate BFS and put the node visitation order on the graph. As in the given example, also show the moves using arrows and operator identifiers (ϕ_0, etc.).

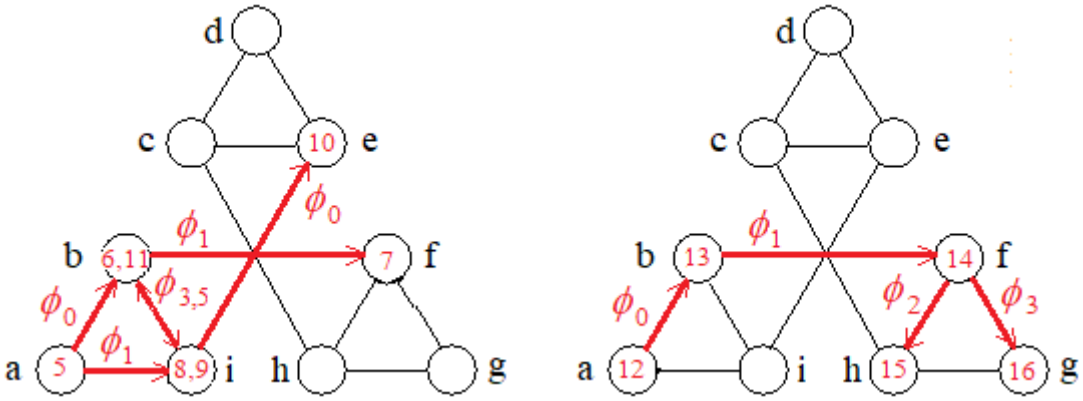


(c) (12 points) Hand-simulate IDDFS and put the node visitation order onto the four graphs below. Use one graph copy for each iteration of IDDFS. The first graph should have only one node (for the initial state) visited. on the graph. The second graph's initial state should have visitation number 2 (since this state is visited again). Note that within one iteration of IDDFS, some node(s) may be reached multiple times along different paths from the initial state. This is OK, and each such repeat visit should be counted as a node visitation and shown in your results. However, there should be no more than one visitation of any given node along the current path between the initial state and current state. That is, the search path must never be allowed to loop back on itself.

First and second iterations:

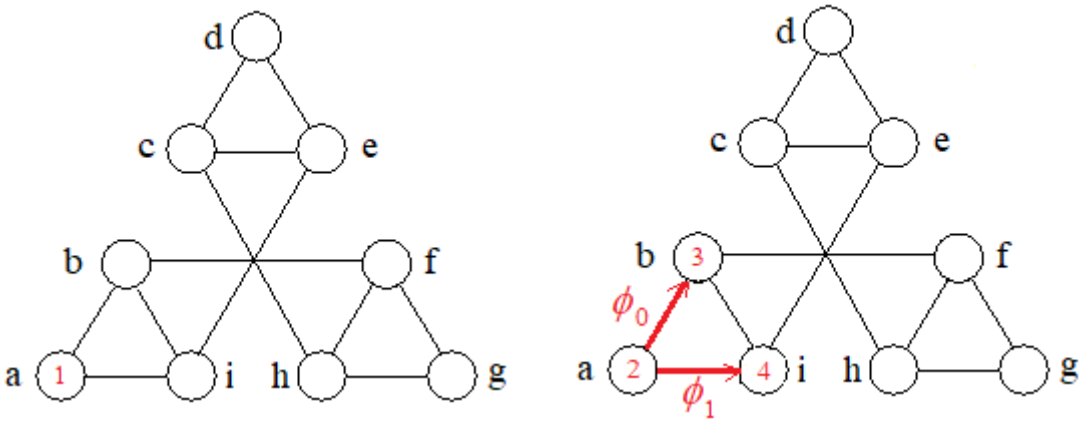


Third and fourth iterations:

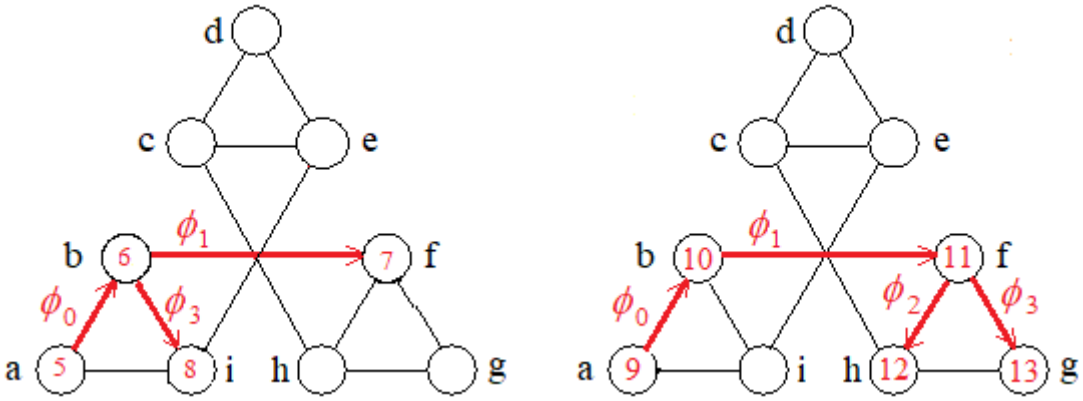


Alternate Solution (in which states are not visited more than once during Iteration 3.)

First and second iterations:



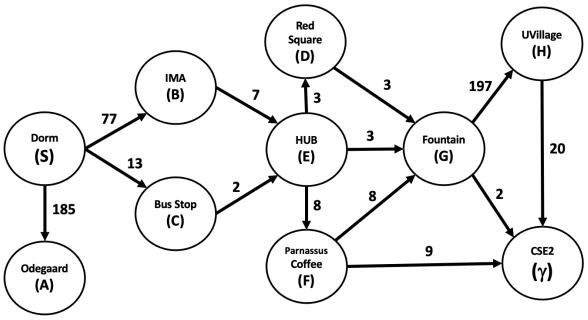
Third and fourth iterations:



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.* Is this always true? Explain why you agree or disagree with the statement. **Hint:** Consider what "closest" does/doesn't guarantee and consider how "best" might be determined or defined.

In selecting a heuristic, you want to choose one that can quickly and easily return a value. If there is too much work involved in calculating your heuristic, the benefit of using a heuristic is lost. Thus, there is generally a trade-off between the ease of calculation of the heuristic and its accuracy. If the heuristic that gives the closest estimate takes too much time to calculate, it can be useless, in which case a somewhat less accurate heuristic could be better. Also, if the estimate closest to the true costs overestimates any cost, the heuristic would not be admissible, which would also make it a worse choice.



state (s)	s_0	A	B	C	D	E	F	G	H	γ
heuristic $h_1(s)$	13	25	11	5	5	4	10	2	15	0
heuristic $h_2(s)$	14	50	12	6	5	5	8	2	18	0
heuristic $h_3(s)$	14	5	5	7	4	1	7	1	19	0

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

An admissible heuristic is one that never overestimates the distance to the goal. An admissible heuristic is not necessarily consistent. In the heuristics given, all the heuristics except h_1 (where h(F) exceeds the actual distance) are admissible, for the others, no heuristic values are overestimating the distance to the goal.

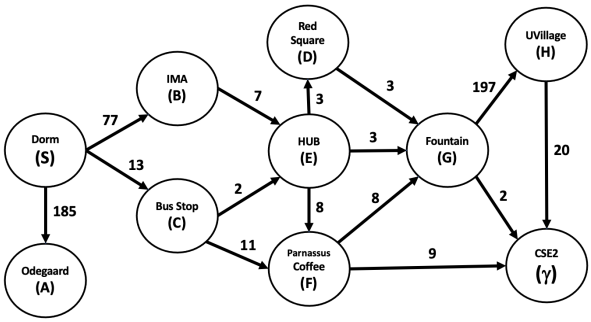
(c) (5 points) Which heuristics (h_1, h_2, h_3) shown above are consistent?

A consistent heuristic is one that is internally consistent in how it calculates the distance to the goal. In other words, if one node is closer to the goal than some other node, this will be reflected in the values of each node's heuristic. Thus, when going from neighboring nodes, the heuristic difference/step cost never overestimates the actual step cost – $(h(N) \leq c(N,P) + h(P))$, where N is some node and P is a successor node of N. A consistent heuristic is also admissible (though the converse is not guaranteed). The calculation can also be written as: $h(N) - h(P) \leq c(N,P)$

In h_1, $h(F) = 10$ and $h(goal) = 0$, while $c(F, goal) = 9$. We see that $h(F) > c(F, goal) + h(goal)$, namely, $10 > 9 + 0$. This means that h_1 is not consistent. Checking other node pairs in similar fashion, we see h_2 is consistent and h_3 is not consistent ($h(C) > c(C,E) + h(E)$ or $7 > 2 + 1$).

(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.

With a non-admissible heuristic, the A algorithm could overlook the optimal solution to a search problem due to an overestimation in $f(n)$. With a non-consistent heuristic, the A* algorithm would require repeated expansion for a node every time a new best (so-far) cost is achieved for it. Clearly, the best heuristic to use would be one that is both admissible AND consistent, so h_2 is our best choice.*



state (s)	s_0	A	B	C	D	E	F	G	H	γ
heuristic $h_4(s)$	18	50	11	6	4	4	8	1	18	0

(e) (4 points) Referring back to the graph (**NOTE:** new edge added – coffee's important!), determine the path that would be computed by an A* search, given the heuristics provided above. As you search the space, complete the table below, indicating which nodes are on the OPEN and CLOSED lists, along with their f values:

See table on the next page (too long to fit below).

	OPEN	CLOSED
Starting A^* search	[s_0,18]	empty
take s_0 off OPEN	empty	[s_0,18]
add successors to OPEN	[C,19],[B,88],[A,235],	[s_0,18]
take [C,19] off OPEN	[B,88],[A,235]	[s_0,18],[C,19]
add successors to OPEN	[E,19],[F,32],[B,88],[A,235]	[s_0,18],[C,19]
take [E,19] off OPEN	[F,32],[B,88],[A,235]	[s_0,18],[C,19],[E,19]
add successors to OPEN	[G,19],[D,22],[F,31],~~[F,32]~~, [B,88],[A,235]	[s_0,18],[C,19],[E,19]
take [G,19] off OPEN	[D,22], [F,31],~~[F,32]~~,[B,88], [A,235]	[s_0,18],[C,19],[E,19],[G,19]
add successors to OPEN	[γ,20],[D,22],[F,31],~~[F,32]~~, [B,88],[H,233],[A,235]	[s_0,18],[C,19],[E,19],[G,19]
take [γ,20] off OPEN	[D,22],[F,31],~~[F,32]~~,[B,88], [H,233],[A,235]	[s_0,18],[C,19],[E,19],[G,19],[γ,20]

GOAL FOUND!

After doing A*, the CLOSED list contains: [s_0,18],[C,19],[E,19],[G,19],[γ,20]

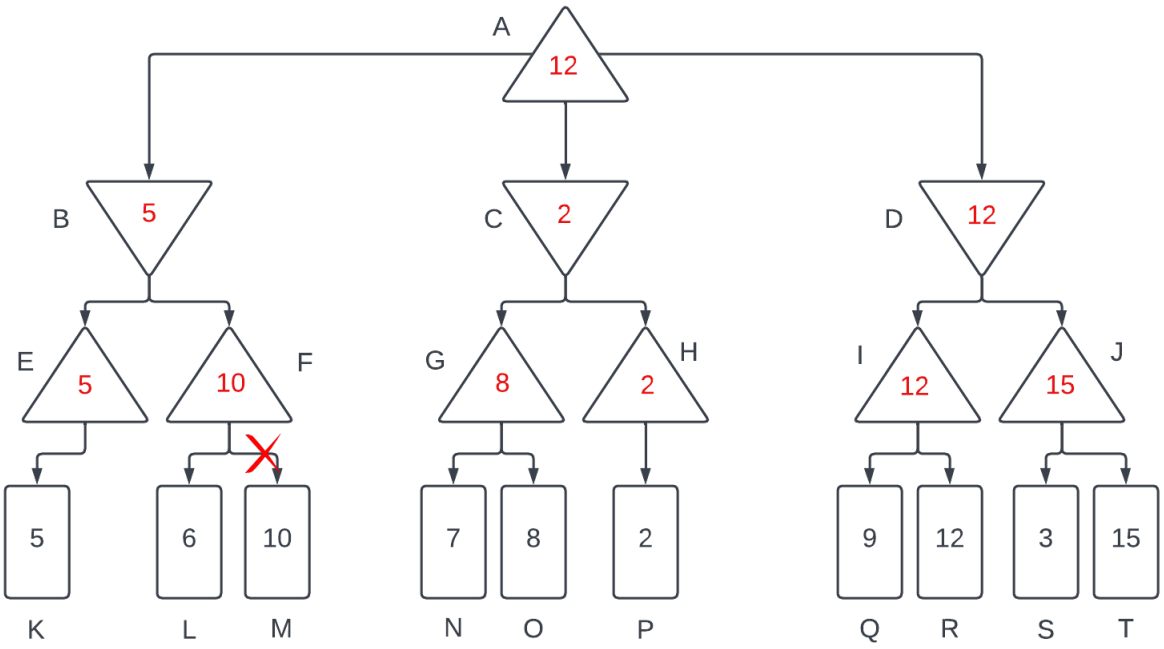
Backtracing from [γ,20], we get: [s_0,18]←[C,19]←[E,19]←[G,19]←[γ,20]

NOTE: Nodes in ~~strikethrough font~~ represent nodes that were replaced by lower-cost versions of themselves.

3 Adversarial Search

Minimax game-tree search finds the best move under the assumption that both players play rationally to respectively maximize and minimize the utility of the same evaluation function. (It can also do well when those assumptions are relaxed.) The Minimax search, however, typically requires a lot of computation time, especially with a deep and widely branching search tree. Alpha-beta pruning is a method for speeding up Minimax search by skipping any subtrees from the search tree that will not contribute to the outcome. Despite that, Alpha-beta pruning's success is dependent upon the order in which we visit the states. A better order to visit the nodes could result in large difference in performance.

For the following questions, note that the example we are using has a maximizing node at the root, which means our goal is to determine which choice of move at A offers the maximum value.



(a) (7 points) Apply straight minimax search, depth-first, left-to-right. Show the resulting values at each internal node. Then fill in the table. The total number of nodes visited should include counts for the root node and the leaf nodes.

Number of nodes visited:	20

(b) (3 points) Apply alpha-beta pruning, left-to-right, on the given tree. Mark where cutoffs occur in the tree above, and fill out the tables. Note: Use the **final** resulting alpha-beta values for the second table. You can also mark these values in the graph if you prefer. You do **NOT** need to fill in the resulting values at each internal node for this part):

Number of nodes visited:	19
Count of cutoffs:	1

Node	A	B	C	D	E	F	G	H	I	J
alpha value	12	-inf	5	5	5	6	8	5	12	15
beta value	inf	5	2	12	inf	5	inf	8	inf	12

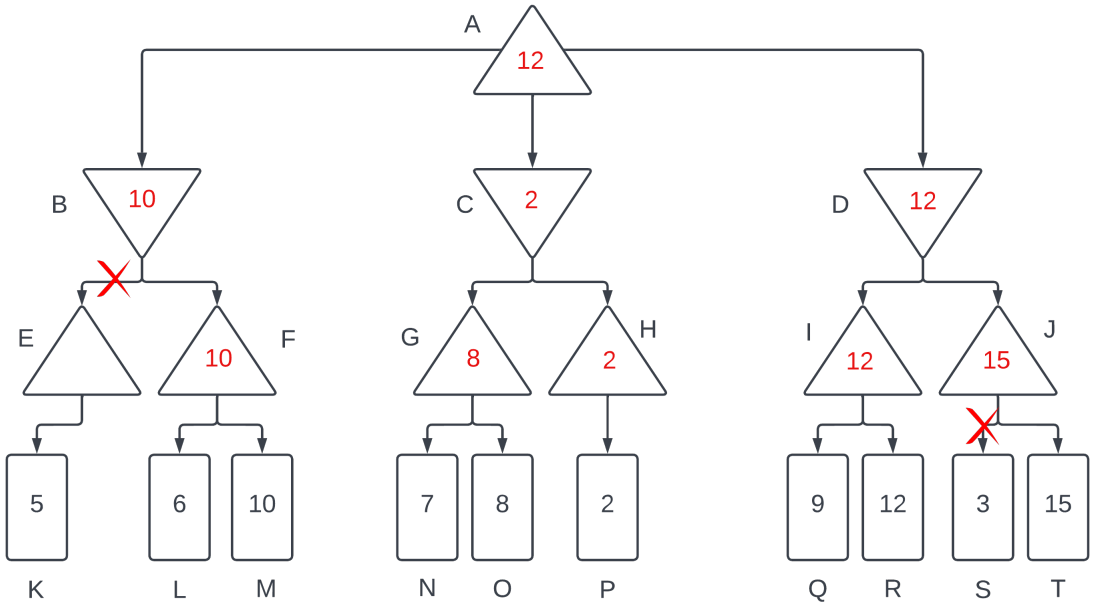
(c) (10 points) Apply alpha-beta pruning, using an auxiliary static evaluation function $f_2(s)$. You are still going depth-first in some sense, but use the auxiliary static evaluation function to determine the order for visiting the children nodes. Starting from the root node, sort each of its children from high to low using f_2. Then visit them following that order. Simiarly, do this for all the non-leaf nodes you visit, except that when visiting a minimizing node, sort its children from low to high. You will still look for cutoffs when applicable.

For this problem, Use f_2 as given in the tables. (In practice, we might invest a lot of resources into computing the auxiliary static evaluation function that would result in a good ordering prior to alpha-beta pruning.)

Node s:	A	B	C	D	E	F	G	H	I	J
$f_2(s)$:	0	1	3	2	2	1	5	6	3	4

Node s:	K	L	M	N	O	P	Q	R	S	T
$f_2(s)$:	3	2	1	9	8	10	4	5	6	7

Show the resulting values at each internal node (leave it blank if pruned), mark where cutoffs occur on the graph below, and fill out the tables (leave the cell blank if pruned):



Number of nodes visited:	16
Count of cutoffs:	2

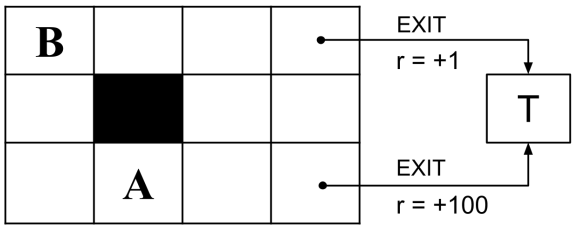
Node	A	B	C	D	E	F	G	H	I	J
alpha value	12	12	-inf	2		12	8	2	12	15
beta value	inf	10	2	12		inf	inf	8	inf	12

4 Markov Decision Processes

Consider the following problem that is taking place in various instances of a Grid-World MDP. The shaded box is walled off and is not one of the possible states. In all states, the agent has available actions $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$. Performing an action that would transition to an invalid state (outside the grid or into a wall) results in the agent remaining in its original state. In states with an arrow coming out, the agent has an *additional* action `EXIT`. In the event that the `EXIT` action is taken, the agent receives the labeled reward and ends the game in the terminal state T. Unless otherwise stated, all other transitions receive no reward, and all transitions are deterministic. (Thus, this MDP is rather different from the Grid-World example shown in lecture, even though they share the same state space.)

For **all** parts of the problem, assume that value iteration begins with all states initialized to zero, i.e., $\forall s \ V_0(s) = 0$. **Let the discount factor be $\gamma = 0.5$ for all the following parts**.

Suppose that we are performing Value Iteration on the Grid-World MDP below:



(a) Fill in the "optimal" values for A and B in the given boxes. (I.e., determine the expected total discounted rewards when starting in A (or B) and following an optimal policy.)

$V^*(A)$: [ ] $100 \cdot 0.5^2 = 25$ $\quad$ $V^*(B)$: [ ] $100 \cdot 0.5^5 = 3.125$ or $\frac{25}{8}$

(b) After how many iterations k will we have $V_k(s) = V^*(s)$ for all states s? If it never occurs, write "never". Write your answer in the given box.

[6] 6 steps to reach B from T

(c) Suppose that we wanted to re-design the reward function. For which of the following new reward functions would the optimal policy **remain unchanged**? Let $R(s, a, s')$ be the original reward function.

[x] $R_1(s, a, s') = 10 \cdot R(s, a, s')$

☒ $R_2(s, a, s') = R(s, a, s') + 1$

☒ $R_3(s, a, s') = R(s, a, s')^2$

☐ $R_4(s, a, s') = -5$

☐ None

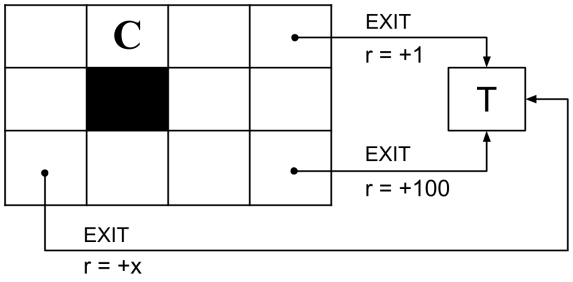
R_1: Scaling the reward function does not affect the optimal policy, as it scales all Q-values by 10, which retains ordering

R_2: Since reward is discounted, the agent would get more reward exiting then infinitely cycling between states

R_3: The only positive reward remains to be from exiting state +100 and +1, so the optimal policy doesn't change

R_4: With negative reward at every step, the agent would want to exit as soon as possible, which means the agent would not always exit at the bottom-right square.

(d) For the following problem, we add a new state in which we can take the `EXIT` action with a reward of $+x$.



(i) For what values of x is it *guaranteed* that our optimal policy π^* has $\pi^*(C) = \leftarrow$ (i.e. going left from C)? Write ∞ and $-\infty$ if there is no definite upper or lower bound, respectively. Write the upper and lower bounds in each respective box.

50	$< x <$	∞

We go left if $Q(C, \leftarrow) > Q(C, \rightarrow)$. $Q(C, \leftarrow) = 0.5^3 x$, and $Q(C, \rightarrow) = 100 \cdot 0.5^4$. Solving for x, we get $x > 50$.

(ii) For what values of x does value iteration take the **minimum** number of iterations k to converge to V^* for all states? Write ∞ and $-\infty$ if there is no upper or lower bound, respectively. Write the upper and lower bounds in each respective box.

50	$\leq x \leq$	200

The two states that will take the longest for value iteration to become non-zero from either $+x$ or $+100$, are states C, and D, where D is defined as the state to the right of C. C will become nonzero at iteration 4 from $+x$, and D will become

nonzero at iteration 4 from +100. We must bound x so that the optimal policy at D does not choose to go to $+x$, or else value iteration will take 5 iterations. Similar reasoning for D and $+x$. Then our inequalities are $0.5^3 x \geq 100 \cdot 0.5^4$ and $0.5^4 x \leq 100 \cdot 0.5^3$. Simplifying, we get the following bound on x: $50 \leq x \leq 200$

(iii) Fill the box with value k, the **minimum** number of iterations until V_k has converged to V^* for all states.

4

See the explanation for the part above: The two states that will take the longest for value iteration to become non-zero from either $+x$ or +100, are states C, and D, where D is defined as the state to the right of C. C will become nonzero at iteration 4 from $+x$, and D will become nonzero at iteration 4 from +100. We must bound x so that the optimal policy at D does not choose to go to $+x$, or else value iteration will take 5 iterations.

5 Computing MDP State Values and Q-Values

Recently, Mike 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 (possibly non-zero) 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 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 symbols of the specific states, e.g., s_1, and actions, e.g., f.)

$$V(s_1) = \max_{a \in \{f,g\}} (R(s_1, a) + \gamma(T(s_1, a, s_2)V(s_2) + T(s_1, a, s_3)V(s_3)))$$
$$V(s_2) = \max_{a \in \{f,g\}} (R(s_2, a) + \gamma(T(s_2, a, s_1)V(s_1) + T(s_2, a, s_3)V(s_3)))$$
$$V(s_3) = \max_{a \in \{f,g\}} (R(s_3, a) + \gamma(T(s_3, a, s_1)V(s_1) + T(s_3, a, s_2)V(s_2)))$$

This is not the only acceptable answer. Alternate definitions that are equally concrete as the ones above will also be acceptable.

(b) (8 points) One fateful day, while Mike 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, Mike 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	-1.1	f
s_1	g	-2.5	-1.1	f
s_2	f	2	2	g
s_2	g	2	2	g
s_3	f	1.41	1.1	f
s_3	g	1.2	1.1	f

After looking at the command line history and noting that a discount of $\gamma = 1$ was specified, Mike 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 recover, rounded to 3 decimal places. HINT: What must transition probabilities out of a given state add up to? Also, how do you relate $V^*(s)$ to the transition probabilities, rewards, and other state values?

s	a	$T(s, a, s_1)$	$T(s, a, s_2)$	$T(s, a, s_3)$
s_1	f	0	0.889	0.111
s_1	g	0	$[0, 0.333]$	$[0.667, 1]$
s_2	f	$[0.500, 1]$	0	$[0, 0.500]$
s_2	g	0.500	0	0.500
s_3	f	0.745	0.255	0
s_3	g	$[0.677, 1]$	$[0, 0.323]$	0

Explanation: We can calculate $T(s, a, s')$ when $a = \pi^*(s)$, because we have the following relationships:

$$V^*(s) = R(s, \pi^*(s)) + T(s, \pi^*(s), s') \cdot V^*(s') + T(s, \pi^*(s), s'') \cdot V^*(s'')$$
$$1 = T(s, \pi^*(s), s') + T(s, \pi^*(s), s'')$$

One can then solve these (2 unknowns, 2 equations) to derive the transition probabilities.
When $a \neq \pi^*(s)$ we don't really know any equality relationships so we cannot calculate the values.
For the action as without values, we do know that it must be the case that the action implies $Q(s, a) \leq Q(s, \pi^*(s)) = V^*(s)$ otherwise the optimal policy should have picked this other a. So:

$$R(s, a) + T(s, a, s')V^*(s') + T(s, a, s'')V^*(s'') \leq V^*(s)$$
$$T(s, a, s') + T(s, a, s'') = 1$$

Note that even here the T's must sum to 1. This means we can solve a bound for both T (just complement of each other). We also note that $0 \leq T \leq 1$ is always the case because they are probabilities, which can be merged together with the new one-sided bound to create a tighter upper/lower bound pair. (Values 4pt, Explanation 4pt)

We use the $[a, b]$ notation here to mean $a \leq x \leq b$. Both notations are acceptable, as is strict less/more-than.