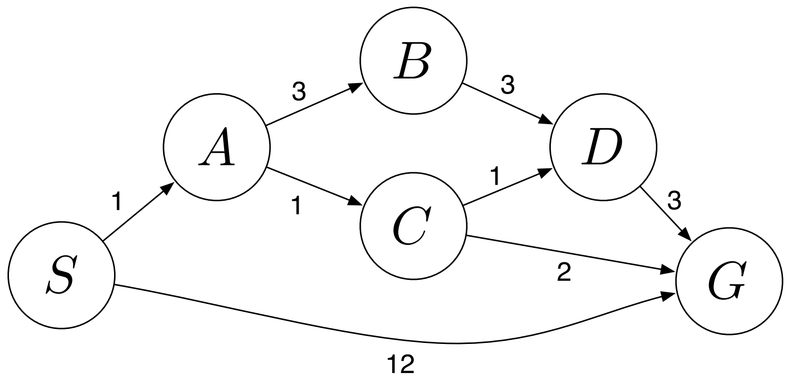
CS188 Fall 2018 Section 6: Midterm 1 Prep

1 . Search



Answer the following questions about the search problem shown above. Assume that ties are broken alphabetically. (For example, a partial plan $S \to X \to A$ would be expanded before $S \to X \to B$; similarly, $S \to A \to Z$ would be expanded before $S \to B \to A$.) For the questions that ask for a path, please give your answers in the form '$S - A - D - G$.'

(a) What path would breadth-first graph search return for this search problem?
$S - G$

(b) What path would uniform cost graph search return for this search problem?
$S - A - C - G$

(c) What path would depth-first graph search return for this search problem?
$S - A - B - D - G$

(d) What path would A* graph search, using a consistent heuristic, return for this search problem?
$S - A - C - G$

(e) Consider the heuristics for this problem shown in the table below.

State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0

(i) Is h_1 admissible? **Yes** [✓ **No**]

(ii) Is h_1 consistent? **Yes** [✓ **No**]

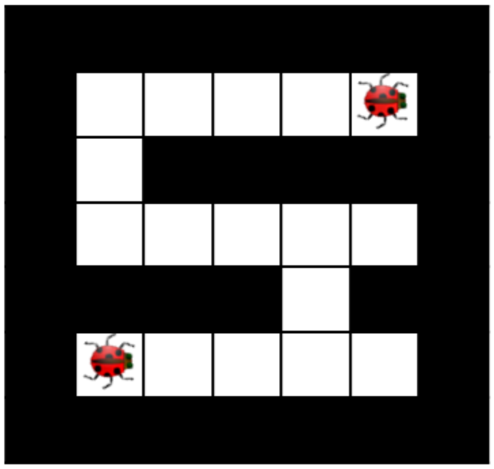
(iii) Is h_2 admissible? [✓ **Yes**] **No**

(iv) Is h_2 consistent? **Yes** [✓ **No**]

2 . Hive Minds: Redux

Let's revisit our bug friends. To recap, you control one or more insects in a rectangular maze-like environment with dimensions $M \times N$, as shown in the figures below. At each time step, an insect can move North, East, South, or West (but not diagonally) into an adjacent square if that square is currently free, or the insect may stay in its current location. Squares may be blocked by walls (as denoted by the black squares), but the map is known.

For the following questions, you should answer for a general instance of the problem, not simply for the example maps shown.



You now control a pair of long lost bug friends. You know the maze, but you do not have any information about which square each bug starts in. You want to help the bugs reunite. You must pose a search problem whose solution is an all-purpose sequence of actions such that, after executing those actions, both bugs will be on the same square, regardless of their initial positions. Any square will do, as the bugs have no goal in mind other than to see each other once again. Both bugs execute the actions mindlessly and do not know whether their moves succeed; if they use an action which would move them in a blocked direction, they will stay where they are. Unlike the flea in the previous question, bugs *cannot* jump onto walls. Both bugs can move in each time step. Every time step that passes has a cost of one.

(a) Give a *minimal* state representation for the above search problem.

A list of boolean variables, one for each position in the maze, indicating whether the position could contain a bug. You don't keep track of each bug separately because you don't know where each one starts; therefore, you need the same set of actions for each bug to ensure that they meet.

(b) Give the size of the state space for this search problem.

2^{MN}

(c) Give a nontrivial admissible heuristic for this search problem.

h_{friends} = the maximum Manhattan distance of all possible pairs of points the bugs can be in.

3 . CSPs: Time Management

Two of our TAs, Arjun and Dave, are making their schedules for a busy morning. There are five tasks to be carried out:

(F) Pick up food for the group's research seminar, which, sadly, takes one precious hour.
(H) Prepare homework questions, which takes 2 consecutive hours.
(P) Prepare the PR2 (robot that Pieter uses for research) for a group of preschoolers' visit, which takes one hour.
(S) Lead the research seminar, which takes one hour.
(T) Teach the preschoolers about the PR2 robot, which takes 2 consecutive hours.

The schedule consists of one-hour slots: 8am-9am, 9am-10am, 10am-11am, 11am-12pm. The requirements for the schedule are as follows:

1. In any given time slot each TA can do at most one task (F, H, P, S, T).
2. The PR2 preparation (P) should happen before teaching the preschoolers (T).
3. The food should be picked up (F) before the seminar (S).
4. The seminar (S) should be finished by 10am.
5. Arjun is going to deal with food pick up (F) since he has a car.
6. The TA not leading the seminar (S) should still attend, and hence cannot perform another task (F, T, P, H) during the seminar.
7. The seminar (S) leader does not teach the preschoolers (T).
8. The TA who teaches the preschoolers (T) must also prepare the PR2 robot (P).
9. Preparing homework questions (H) takes 2 consecutive hours, and hence should start at or before 10am.
10. Teaching the preschoolers (T) takes 2 consecutive hours, and hence should start at or before 10am.

To formalize this problem as a CSP, use the variables F, H, P, S and T. The values they take on indicate the TA responsible for it, and the starting time slot during which the task is carried out (for a task that spans 2 hours, the variable represents the starting time, but keep in mind that the TA will be occupied for the next hour also - make sure you enforce constraint (a)!). Hence there are eight possible values for each variable, which we will denote by A8, A9, A10, A11, D8, D9, D10, D11, where the letter corresponds to the TA and the number corresponds to the time slot. For example, assigning the value of A8 to a variables means that this task is carried about by Arjun from 8am to 9am.

(a) What is the size of the state space for this CSP?

8^5.

(b) Which of the statements above include unary constraints?

(d), (e), (i), (j). (i) and (j) are both unary constraints, and binary constraints in a single sentence.

(c) In the table below, enforce all unary constraints by crossing out values in the table on the left below. If you made a mistake, cross out the whole table and use the right one.

F	A8	A9	A10	A11	~~D8~~	~~D9~~	~~D10~~	~~D11~~
H	A8	A9	A10	~~A11~~	D8	D9	D10	~~D11~~
P	A8	A9	A10	A11	D8	D9	D10	D11
S	A8	A9	~~A10~~	~~A11~~	D8	D9	~~D10~~	~~D11~~
T	A8	A9	A10	~~A11~~	D8	D9	D10	~~D11~~

(d) Start from the table above, select the variable S and assign the value A9 to it. Perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~D8~~	~~D9~~	D10	~~D11~~
P	A8	~~A9~~	A10	A11	D8	~~D9~~	D10	D11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	D10	~~D11~~

(e) Based on the result of (d), what variable will we choose to assign next based on the MRV heuristic (breaking ties alphabetically)? Assign the first possible value to this variable, and perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

Variable F is selected and gets assigned value A8.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~D8~~	~~D9~~	D10	~~D11~~
P	~~A8~~	~~A9~~	A10	A11	D8	~~D9~~	D10	D11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	D10	~~D11~~

Have we arrived at a dead end (i.e., has any of the domains become empty)?

No.

(f) We return to the result from enforcing just the unary constraints, which we did in (c). Select the variable S and assign the value A9. Enforce arc consistency by crossing out values in the table below.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
P	~~A8~~	~~A9~~	~~A10~~	~~A11~~	D8	~~D9~~	~~D10~~	~~D11~~
S	~~A8~~	A9	~~A10~~	~~A11~~	~~D8~~	~~D9~~	~~D10~~	~~D11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~D8~~	~~D9~~	D10	~~D11~~

(g) Compare your answers to (d) and to (f). Does arc consistency remove more values or less values than forward checking does? Explain why.

Arc consistency removes more values. It's because AC checks consistency between any pair of variables, while FC only checks the relationship between pairs of assigned and unassigned variables.

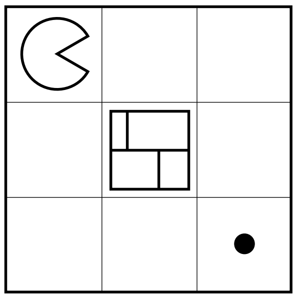
(h) Check your answer to (f). Without backtracking, does any solution exist along this path? Provide the solution(s) or state that there is none.

AC along this path gives 1 solution: F: A8 H: A10 P: D8 S: A9 T: D10

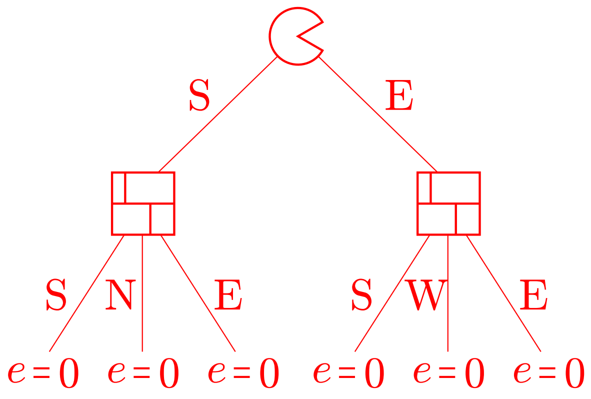
4 . Surrealist Pacman

In the game of Surrealist Pacman, Pacman plays against a moving wall . On Pacman's turn, Pacman must move in one of the four cardinal directions, and must move into an unoccupied square. On the wall's turn, the wall must move in one of the four cardinal directions, and must move into an unoccupied square. The wall cannot move into a dot-containing square. Staying still is not allowed by either player. Pacman's score is always equal to the number of dots he has eaten.

The first game begins in the configuration shown below. Pacman moves first.



(a) Draw a game tree with one move for each player. Nodes in the tree represent game states (location of all agents and walls). Edges in the tree connect successor states to their parent states. Draw only the legal moves.



(b) According to the depth-limited game tree you drew above what is the value of the game? Use Pacman's score as your evaluation function.

0. All leaves have value 0.

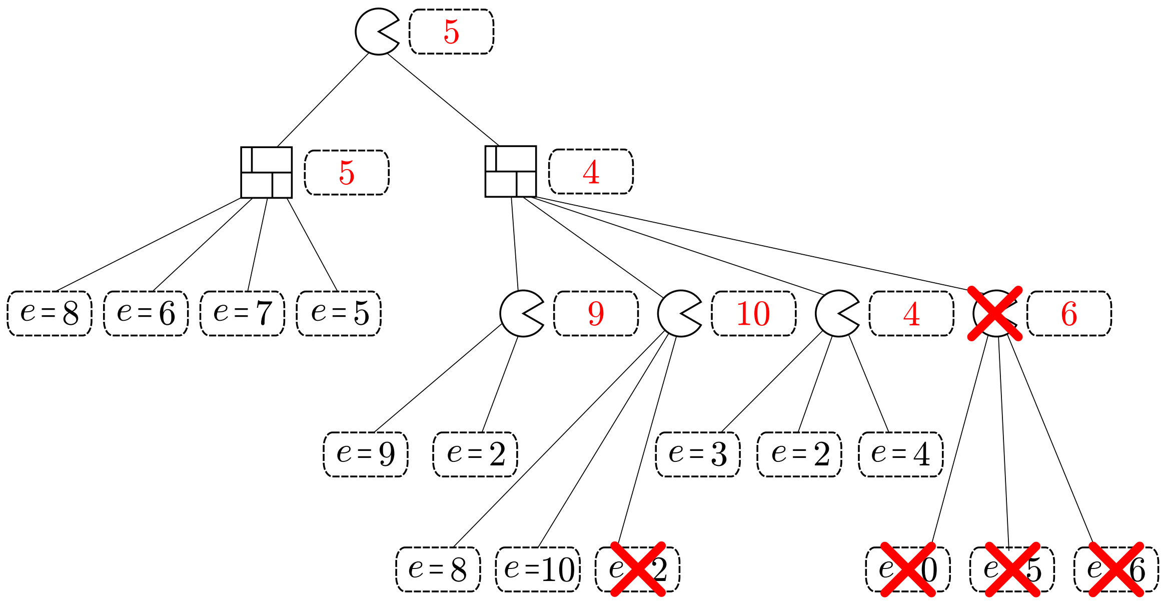
(c) If we were to consider a game tree with ten moves for each player (rather than just one), what would be the value of the game as computed by minimax?

1. Pacman can force a win in ten moves.

A second game is played on a more complicated board. A partial game tree is drawn, and leaf nodes have been scored using an (unknown) evaluation function e.

(d) In the dashed boxes, fill in the values of all internal nodes using the minimax algorithm.

(e) Cross off any nodes that are not evaluated when using alpha-beta pruning (assuming the standard left-to-right traversal of the tree).



Running alpha-beta pruning on the game tree.

Root: $\alpha = -\infty, \beta = \infty$

- - Left wall: $\alpha = -\infty, \beta = \infty$

- - - - Leaf node: e = 8. Propagate e = 8 back to parent.

- - Left wall: Current value is 8. $\alpha = -\infty, \beta = 8$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 6. Propagate e = 6 back to parent.

- - Left wall: Current value is 6. $\alpha = -\infty, \beta = 6$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 7. Propagate e = 7 back to parent.

- - Left wall: No update. Current value is 6. $\alpha = -\infty, \beta = 6$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 5. Propagate e = 5 back to parent.

- - Left wall: Current value is 5. We're done here, so propagate 5 to root.

Root: Current value is 5. $\alpha = 5, \beta = \infty$. Explore right.

- - Right wall: $\alpha = 5, \beta = \infty$.

- - - - 1st Pac: $\alpha = 5, \beta = \infty$

- - - - - - Leaf node: e = 9. Propagate e = 9 back to parent.

- - - - 1st Pac: Current value is 9. $\alpha = 9, \beta = \infty$ MIN doesn't have a best value, so continue exploring.

- - - - - - Leaf node: e = 2. Propagate e = 2 back to parent.

- - - - 1st Pac: No change. Current value is 9. Propagate 9 to parent.

- - Right wall: Current value is now 9. $\alpha = 5, \beta = 9$. MIN wants anything less than 9 at this point, but it's still possible for MAX to get more than 5. Continue exploring.

- - - - 2nd Pac: $\alpha = 5, \beta = 9$

- - - - - - Leaf node: e = 8. Propagate e = 8 back to parent.

- - - - 2nd Pac: Current value is now 8. $\alpha = 8, \beta = 9$. Again, still possible for both players to benefit (Imagine value = 8.5). Continue exploring.

- - - - - - Leaf node: e = 10. Propagate e = 10 back to parent.

- - - - 2nd Pac: Current value is now 10. So now, we know that $v > \beta$, which means that one of the players is going to be unhappy. MAX wants something more than 10, but MIN is only satisfied with something less than 9, so we don't have to keep exploring.

- - - - *PRUNE e = 2.*

- - - - 2nd Pac: returns value of 10 to parent.

- - Left Wall: No change in value, current value is still 9. $\alpha = 5, \beta = 9$. Again, still possible for both players to benefit, so continue exploring.

- - - - 3rd Pac: $\alpha = 5, \beta = 9$

- - - - - - Leaf node: e = 3. Propagate e = 3 back to parent.

- - - - 3rd Pac: Current value is 3. $\alpha = 5, \beta = 9$. Continue exploring.

- - - - - - Leaf node: e = 2. Propagate e = 2 back to parent.

- - - - 3rd Pac: No change in value. Current value is 3. $\alpha = 5, \beta = 9$. Continue exploring.

- - - - - - Leaf node: e = 4. Propagate e = 4 back to parent.

- - - - 3rd Pac: Current value is 4. We're done, so return value of 4 to parent.

- - Left Wall: Current value becomes 4. At this point, we know that MIN wants anything that is less than or equal to 4. However, MAX is only satisfied with something that is 5 or greater. Hence, we don't need to explore the rest of the children of this node since MAX will never let the game get down to this branch.- -

Prune rest

(Filling values returned by alpha-beta or not crossing off children of a crossed off node were not penalized.)

Suppose that this evaluation function has a special property: it is known to give the correct minimax value of any internal node to within 2, and the correct minimax values of the leaf nodes exactly. That is, if v is the true minimax value of a particular node, and e is the value of the evaluation function applied to that node, $e - 2 \le v \le e + 2$, and $v = e$ if the node is a dashed box in the tree below.

Using this special property, you can modify the alpha-beta pruning algorithm to prune more nodes.

(f) Standard alpha-beta pseudocode is given below (only the max-value recursion). Fill in the boxes on the right to replace the corresponding boxes on the left so that the pseudocode prunes as many nodes as possible, taking account of this special property of the evaluation function.

```
function MAX-VALUE(node, α, β)
    e ← EVALUATIONFUNCTION(node)
    if node is leaf then
        return e
    end if
    [          ] (1)
    v ← −∞
    for child ← CHILDREN(node) do
        [ v ← MAX(v, MIN-VALUE(child, α, β)) ] (2)
        if v ≥ β then
            return v
        end if
        α ← MAX(α, v)
    end for
    return v
end function
```

Fill in these boxes:

(1)

```
    if $e - 2 \ge \beta$ then
        return $e - 2$
    end if
```

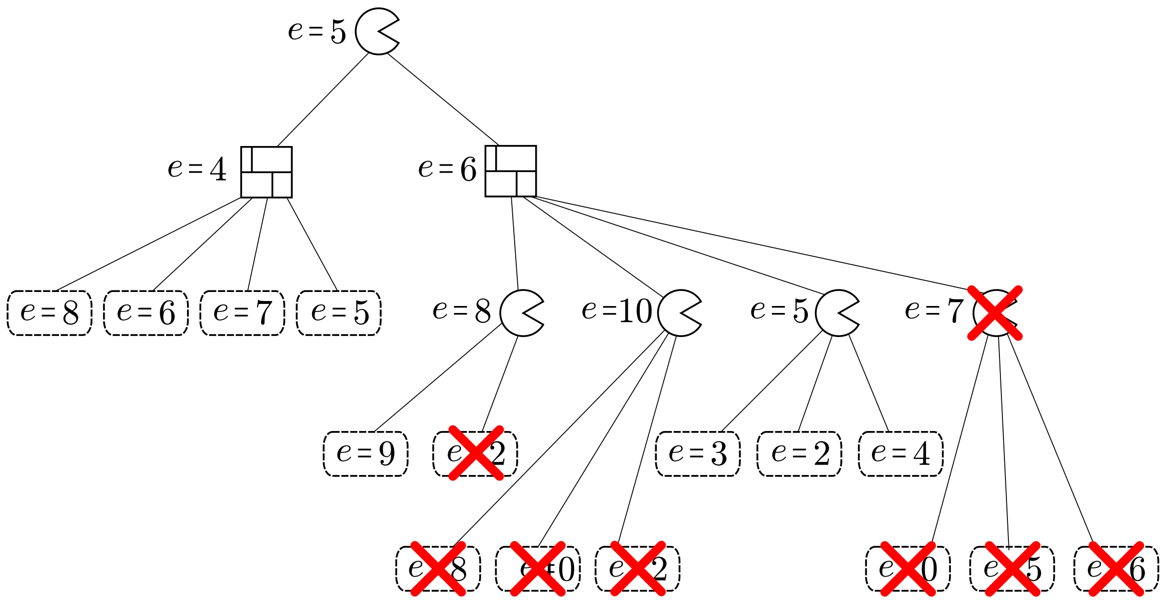
(2)

```
    $v \leftarrow$ MAX($v$, MIN-VALUE($child$,
    MAX($\alpha, e - 2$), MIN($\beta, e + 2$)))
```

(Variations are possible.)

The same game tree is shown below, with the evaluation function applied to *internal* as well as leaf nodes.

(g) In the game tree below cross off any nodes that can be pruned assuming the special property holds true. If not sure you correctly formalized into pseudo-code your intuition on how to exploit the special property for improved pruning, make sure to annotate your pruned nodes with a brief explanation of why each of them was pruned.



Running pruning on game tree in detail. W1 and W2 refer to the left and right wall nodes respectively. P1, P2, P3, P4 refer to Pacman nodes on the third level, left to right.

Root: Evaluation function returns 5. Range of value: [3, 7]. Set $\alpha : 3, \beta : 7$ and explore(W1, α, β).

- W1: Evaluation function returns 4. Range of value: [2,6]. Set $\alpha : 3, \beta : 6$.

- - Leaf node: $e = 8$. Send 8 back to W1.

- W1: $v = 8$. $v < \alpha$? No. This means that MAX might still prefer this path.

- - Leaf node: $e = 6$. Send 6 back to W1.

- W1: $6 < 8$ so $v = 6$. $v < \alpha$? No. Continue exploring.

- - Leaf node: $e = 7$. Send 7 back to W1.

- W1: $7 > 6$, so no change, $v = 6$. $v < \alpha$? No. Continue exploring.

- - Leaf node: $e = 5$. Send 5 back to W1.

- W1: $5 < 6$, so $v = 5$. Done. Send 5 back to root.

Root: Gets value 5, so $v = 5$. $\alpha : \max(3, 5) = 5, \beta : 7$. Still exploring right branch since MAX could get a value $5 \leq v \leq 7$. Explore(W2, α, β).

- W2: Evaluation function returns 6. Range of values $[4, 8]$. $\alpha : 5, \beta : 7$, explore(P1,α, β).

- - P1: Evaluation function returns 8. Range: $[6, 10]$, $\alpha : 6, \beta : 7$.

- - - Leaf node: $e = 9$. Send $e = 9$ back to P1.

- - P1: $v = 9$. $v > \beta$? *Yes!*. We can prune here since P1 wants any value > 9. However, at the root we know that the maximum value that MAX can get is 7. Hence, there is no way the game can get down to P1. (Meaning that the value at the root can not be 9).

- - - Leaf node: Prune $e = 2$. Return 9 to W2.

- W2: $v = 9$. α, β don't change: $\alpha : 5, \beta : 7$. Explore(P2, α, β).

- - P2: Evaluation function returns 10. Range $[8, 12]$, $\alpha : 8, \beta : 7$. Notice that the best value for MIN that can be achieved at P2 is 8. However, the best value for MIN at the root is 7. Hence, there's no way the game can get down to P2. (Meaning the value of the root can not by 8). *Prune all of P2's children!*. We can return 8 to W2 since we know that there is some other path through W2 that yields a reward ≤ 7.

- W2: $v : \min(8, 9) = 8, \alpha : 5, \beta : 7$. $v < \alpha$? No! Explore(P3,α, β).

- - P3: Evaluation function returns 5. Range: $[3, 7], \alpha : 5, \beta : 7$.

- - - Leaf node: $e = 5$. Send 3 back to P3.

- - P3: $v = 3.v > \beta$? No! Meaning MIN might still prefer this branch. $\alpha : 5, \beta : 7$.

- - - Leaf node: $e = 2$. Send 2 back to P3.

- - P3: $v = 3.v > \beta$? No! $\alpha : 5, \beta : 7$.

- - - Leaf node: $e = 4$. Send 4 back to P3.

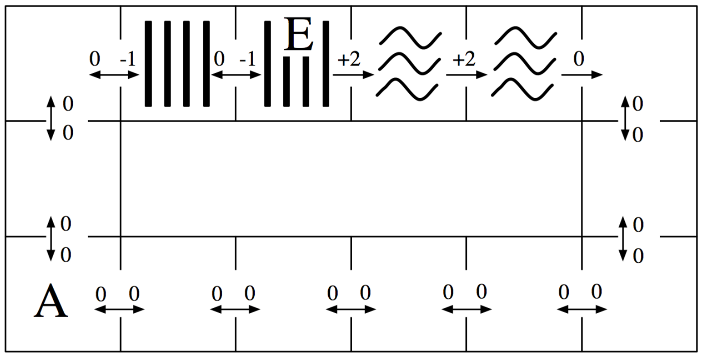
- - P3: $v = 4$. Done. Return 4 to W2.

- W2: $v : \min(8, 4) = 4, \alpha : 5, \beta : 7$. $v < \alpha$? *Yes!* Since MAX can guarantee a value of 5 and MIN will only accept something < 4, don't need to explore any further. *Prune P4 and all its children.* Return 4 to root.

Root: *Done.*

5 . MDPs: Grid-World Water Park

Consider the MDP drawn below. The state space consists of all squares in a grid-world water park. There is a single waterslide that is composed of two ladder squares and two slide squares (marked with vertical bars and squiggly lines respectively). An agent in this water park can move from any square to any neighboring square, unless the current square is a slide in which case it must move forward one square along the slide. The actions are denoted by arrows between squares on the map and all deterministically move the agent in the given direction. The agent cannot stand still: it must move on each time step. Rewards are also shown below: the agent feels great pleasure as it slides down the water slide (+2), a certain amount of discomfort as it climbs the rungs of the ladder (-1), and receives rewards of 0 otherwise. The time horizon is infinite; this MDP goes on forever.



(a) How many (deterministic) policies π are possible for this MDP?

2^{11}

(b) Fill in the blank cells of this table with values that are correct for the corresponding function, discount, and state. *Hint: You should not need to do substantial calculation here.*

	γ	$s = A$	$s = E$
$V_3^*(s)$	1.0	0	4
$V_{10}^*(s)$	1.0	2	4
$V_{10}^*(s)$	0.1	0	2.2
$Q_1^*(s, \text{west})$	1.0	——	0
$Q_{10}^*(s, \text{west})$	1.0	——	3
$V^*(s)$	1.0	∞	∞
$V^*(s)$	0.1	0	2.2

$V_{10}^*(A), \gamma = 1$: In 10 time steps with no discounting, the rewards don't decay, so the optimal strategy is to climb the two stairs (-1 reward each), and then slide down the two slide squares (+2 rewards each). You only have time to do this once. Summing this up, we get $-1 - 1 + 2 + 2 = 2$.

$V_{10}^*(E), \gamma = 1$: No discounting, so optimal strategy is sliding down the slide. That's all you have time for. Sum of rewards $= 2 + 2 = 4$.

$V_{10}^*(A), \gamma = 0.1$. The discount rate is 0.1, meaning that rewards 1 step further into the future are discounted by a factor of 0.1. Let's assume from A, we went for the slide. Then, we would have to take the actions $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E, E \rightarrow F, F \rightarrow G$. We get the first -1 reward from $C \rightarrow D$, discounted by γ^2 since it is two actions in the future. $D \rightarrow E$ is discounted by γ^3, $E \rightarrow F$ by γ^4, and $F \rightarrow G$ by γ^5. Since γ is low, the positive rewards you get from the slide have less of an effect as the larger negative rewards you get from climbing up. Hence, the sum of rewards of taking the slide path would be negative; the optimal value is 0.

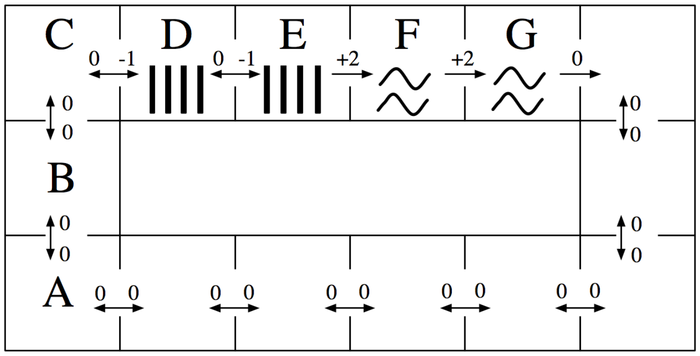
$V_{10}^*(E), \gamma = 0.1$. Now, you don't have to do the work of climbing up the stairs, and you just take the slide down. Sum of rewards would be 2 (for $E \rightarrow F$) + 0.2 (for $F \rightarrow G$, discounted by 0.1) = 2.2.

$Q_{10}^*(E, west), \gamma = 1$. Remember that a Q-state (s,a) is when you start from state s and are committed to taking a. Hence, from E, you take the action West and land in D, using up one time step and getting an immediate reward of 0. From D, the optimal strategy is to climb back up the higher flight of stairs and then slide down the slide. Hence, the rewards would be $-1(D \rightarrow E) + 2(E \rightarrow F) + 2(F \rightarrow G) = 3$.

$V^*(s), \gamma = 1$. Infinite game with no discount? Have fun sliding down the slide to your content from anywhere.

$V^*(s), \gamma = 0.1$. Same reasoning apply to both A and E from $V_{10}^*(s)$. With discounting, the stairs are more costly to climb than the reward you get from sliding down the water slide. Hence, at A, you wouldn't want to head to the slide. From E, since you are already at the top of the slide, you should just slide down.

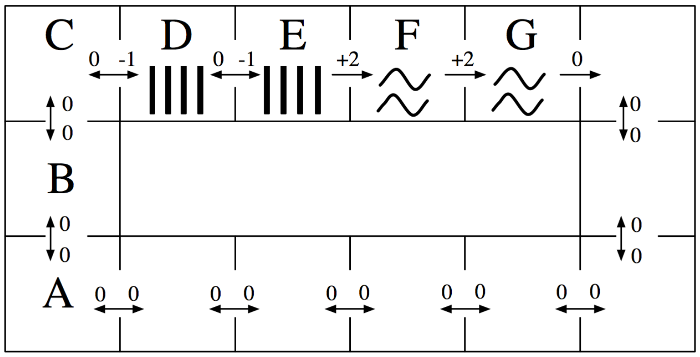
Use this labeling of the state space to complete the remaining subproblems:



(c) Fill in the blank cells of this table with the Q-values that result from applying the Q-update for the transition specified on each row. You may leave Q-values that are unaffected by the current update blank. Use discount $\gamma = 1.0$ and learning rate $\alpha = 0.5$. Assume all Q-values are initialized to 0. (Note: the specified transitions would not arise from a single episode.)

	$Q(D, \text{west})$	$Q(D, \text{east})$	$Q(E, \text{west})$	$Q(E, \text{east})$
Initial:	0	0	0	0
Transition 1: $(s = D, a = \text{east}, r = -1, s' = E)$		-0.5		
Transition 2: $(s = E, a = \text{east}, r = +2, s' = F)$				1.0
Transition 3: $(s = E, a = \text{west}, r = 0, s' = D)$				
Transition 4: $(s = D, a = \text{east}, r = -1, s' = E)$		-0.25		

The agent is still at the water park MDP, but now we're going to use function approximation to represent Q-values. Recall that a policy π is *greedy* with respect to a set of Q-values as long as $\forall a, s\ Q(s, \pi(s)) \geq Q(s, a)$ (so ties may be broken in any way).



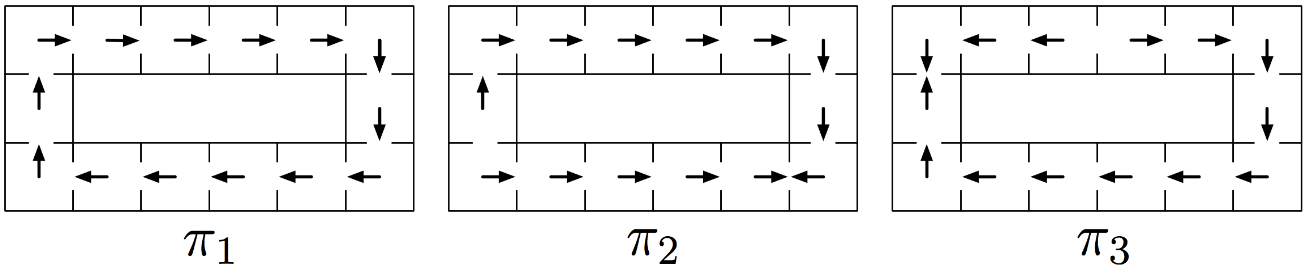
For the next subproblem, consider the following feature functions:

$$f(s,a) = \begin{cases} 1 & \text{if } a = \text{east}, \\ 0 & \text{otherwise.} \end{cases}$$

$$f'(s,a) = \begin{cases} 1 & \text{if } (a = \text{east}) \wedge \text{isSlide}(s), \\ 0 & \text{otherwise.} \end{cases}$$

(Note: isSlide(s) is true iff the state s is a slide square, i.e. either F or G.)

Also consider the following policies:



(d) Which are greedy policies with respect to the Q-value approximation function obtained by running the single Q-update for the transition $(s = F, a = \text{east}, r = +2, s' = G)$ while using the specified feature function? You may assume that all feature weights are zero before the update. Use discount $\gamma = 1.0$ and learning rate $\alpha = 1.0$. Circle all that apply.

f	π_1	[✓ π_2]	π_3
f'	[✓ π_1]	[✓ π_2]	[✓ π_3]

You see the sample (F, east, G, +2). Use approximate Q-Learning to update the weights.

You should get that the new weights are both going to be positive since the sample reward was positive and the feature value was on for both $f(F, east)$ [since you took action east] and $f'(F, east)$ [since you took action east, and you were on the water slide].

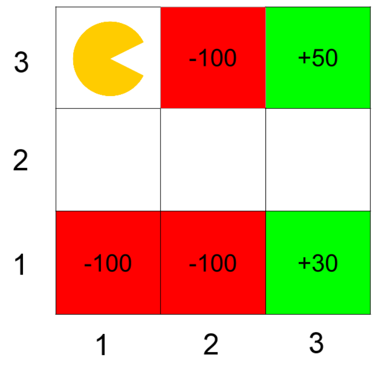
Now, with your new weights, you need to see which greedy policy can be possible.

For f, going East is preferred if possible (when you calculate the Q-value, any Q-state with action east has a positive value, anything else has a value of 0. Hence, throw out π_1 and π_3, since some arrows go west.

For f', going East is preferred *if* you are on the slide (otherwise, everything else is just 0). All three policies contain the fact that you move east from F and G, so all policies are good.

6 . Deep inside Q-learning

Consider the grid-world given below and an agent who is trying to learn the optimal policy. Rewards are only awarded for taking the *Exit* action from one of the shaded states. Taking this action moves the agent to the Done state, and the MDP terminates. Assume $\gamma = 1$ and $\alpha = 0.5$ for all calculations. All equations need to explicitly mention γ and α if necessary.



(a) The agent starts from the top left corner and you are given the following episodes from runs of the agent through this grid-world. Each line in an Episode is a tuple containing (s, a, s', r).

Episode 1	Episode 2	Episode 3	Episode 4	Episode 5
(1,3), S, (1,2), 0	(1,3), S, (1,2), 0	(1,3), S, (1,2), 0	(1,3), S, (1,2), 0	(1,3), S, (1,2), 0
(1,2), E, (2,2), 0	(1,2), E, (2,2), 0	(1,2), E, (2,2), 0	(1,2), E, (2,2), 0	(1,2), E, (2,2), 0
(2,2), E, (3,2), 0	(2,2), S, (2,1), 0	(2,2), E, (3,2), 0	(2,2), E, (3,2), 0	(2,2), E, (3,2), 0
(3,2), N, (3,3), 0	(2,1), Exit, D, -100	(3,2), S, (3,1), 0	(3,2), N, (3,3), 0	(3,2), S, (3,1), 0
(3,3), Exit, D, +50		(3,1), Exit, D, +30	(3,3), Exit, D, +50	(3,1), Exit, D, +30

Fill in the following Q-values obtained from direct evaluation from the samples:

Q((3,2), N) = 50 Q((3,2), S) = 30 Q((2,2), E) = 40

Direct evaluation is just averaging the discounted reward after performing action a in state s.

(b) Q-learning is an online algorithm to learn optimal Q-values in an MDP with unknown rewards and transition function. The update equation is:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a'))$$

where γ is the discount factor, α is the learning rate and the sequence of observations are $(\cdots, s_t, a_t, s_{t+1}, r_t, \cdots)$. Given the episodes in (a), fill in the time at which the following Q values first become non-zero. Your answer should be of the form (**episode#,iter#**) where **iter#** is the Q-learning update iteration in that episode. If the specified Q value never becomes non-zero, write *never*.

Q((1,2), E) = Never Q((2,2), E) = (5,3) Q((3,2), S) = (5,4)

This question was intended to demonstrate the way in which Q-values propagate through the state space. Q-learning is run in the following order - observations in ep 1 then observations in ep 2 and so on.

(c) In Q-learning, we look at a window of (s_t, a_t, s_{t+1}, r_t) to update our Q-values. One can think of using an update rule that uses a larger window to update these values. Give an update rule for $Q(s_t, a_t)$ given the window $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$.

$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2}, a'))$
(Sample of the expected discounted reward using r_{t+1})

$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma((1 - \alpha)Q(s_{t+1}, a_{t+1}) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+2}, a'))))$
(Nested Q-learning update)

$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t)+\alpha(r_t+\gamma \max((1-\alpha)Q(s_{t+1}, a_{t+1})+\alpha(r_{t+1}+\gamma \max_{a'} Q(s_{t+2}, a')), \max_{a'} Q(s_{t+1}, a')))$
(Max of normal Q-learning update and one step look-ahead update)