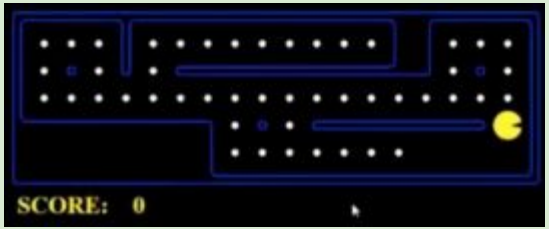
CS 188 Midterm Review Session #1

Pranav Muralikrishnan, Saathvik Selvan, Erin Tan
Adapted from Marwa Abdulhai, Joy Liu, and Kenny Wang
Mon Oct 14, 2024

—-- Search --—

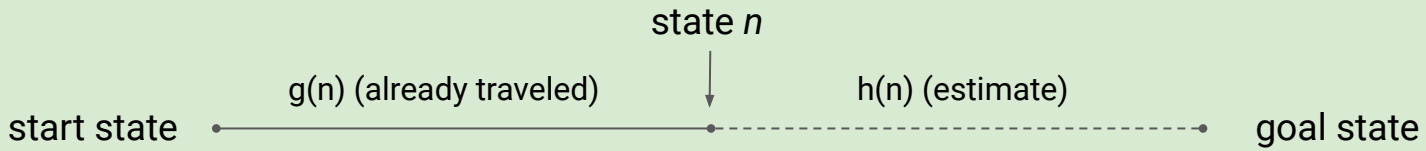
Search Problems

- **State space**: Set of all possible states in the world
- Example: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location
  - Goal state: if (x,y) == END
- Example: Eating all dots
  - States: (x,y) location, dot booleans
  - Actions: NSEW
  - Successor: update location & dot booleans
  - Goal state: dot boolean == false for all dots



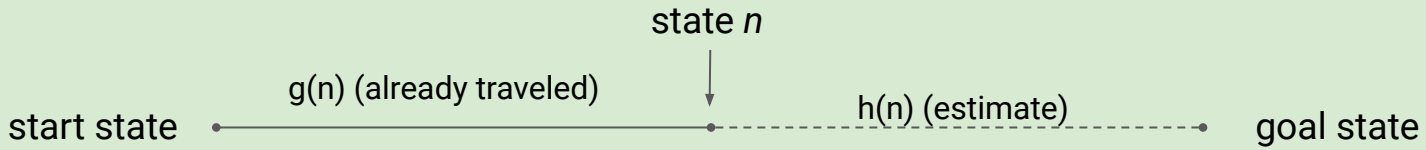
Informed Search (A*)

- Frontier is represented as a priority queue keyed by “lowest estimated total cost” denoted as f(n)
  - f(n) = g(n) + h(n)
    - g(n) = total backwards cost
    - h(n) = estimated forward cost (heuristic)



Informed Search (A*)

- Frontier is represented as a priority queue keyed by “lowest estimated total cost” denoted as f(n)
  - f(n) = g(n) + h(n)
    - g(n) = total backwards cost
    - h(n) = estimated forward cost (heuristic)



- Similar to uniform cost search but with an additional term
- Heuristic is motivated by having prior knowledge about the world
  - example: if we want to drive from New York to San Francisco, we know to generally prioritize exploring routes directed westward

Evaluating Heuristics

h*(n) = true cost till goal from n

Admissibility

- heuristic underestimates the total distance from any given node to the goal
- “optimistic”

$$\forall n,\ 0 \leq h(n) \leq h^*(n)$$

Evaluating Heuristics

h*(n) = true cost till goal from n

Admissibility

- heuristic underestimates the total distance from any given node to the goal
- “optimistic”

$$\forall n,\ 0 \leq h(n) \leq h^*(n)$$

Consistency

- heuristic underestimates the cost/weight of each edge in the graph
- think triangle inequality

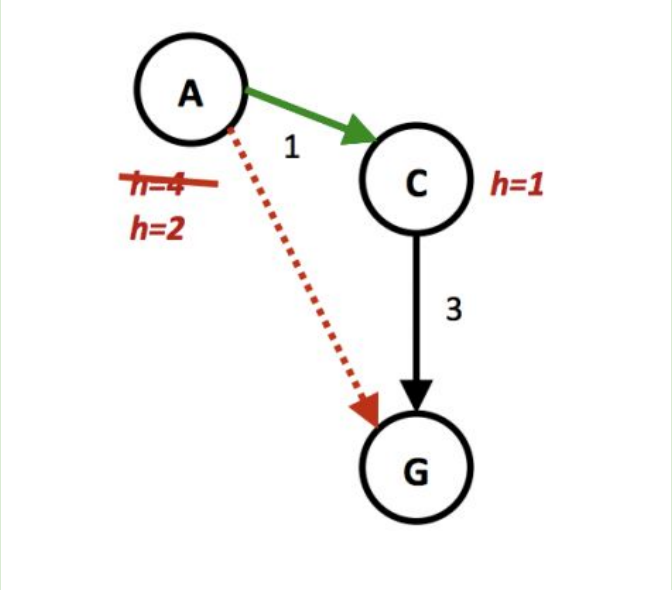
$$\forall A, C \quad h(A) - h(C) \leq cost(A, C)$$

Intuition for Consistency

- the amount our heuristic changes between states should not differ by more than the cost of traveling between the two states

- when h(A) = 4 and h(C) = 1, this suggests that our heuristic estimates the cost of A -> C to be h(A) - h(C) = 3, which is an overestimate (true cost(A, C) = 1)

- triangle inequality: h(A) ≤ h(C) + cost(A, C) or *h(A) - h(C) ≤ cost(A, C)*



Summary

- **Heuristic**: estimate of cost from node n to goal state
  - h(n) = heuristic
- **A* search**: search algorithm where data structure is a priority queue
  - priority f(n) = g(n) + h(n)
  - g(n) = backwards cost
- **Admissible**: heuristic is an underestimate of the true cost
  - $\forall$ n, $0 \leq h(n) \leq h^*(n)$
  - $h^*(n)$ = true cost
  - Necessary for optimality

Q1: Ants: Escape!

An ant wakes up and finds itself in a spider's maze!

- The maze is an M-by-N rectangle.
- Legal actions: $\{Forward, TurnLeft, TurnRight\}$.
- Transition model: *Forward* moves the ant 1 square in the direction it's facing, unless there is a wall in front. The two turning actions rotate the ant by 90 degrees to face a different direction.
- Action cost: Each action costs 1.
- Start state: The ant starts at (s_x, s_y) facing North.
- Goal test: Returns true when the ant reaches the exit at $G = (g_x, g_y)$.

(a) **(i)** What's the minimum state space size S for this task?

$S =$ [ ]

(ii) Now suppose there are K ants, where each ant i must reach a distinct goal location G_i; any number of ants can occupy the same square; and **action costs are a sum of the individual ants' step costs**. What's the minimum state space size for this task, expressed in terms of K and S?

[ ]

Q1: Ants: Escape!

(a) **(i)** What's the minimum state space size S for this task?

$S =$ $4MN$

4 is for storing the direction, MN is for storing the position.

(ii) Now suppose there are K ants, where each ant i must reach a distinct goal location G_i; any number of ants can occupy the same square; and **action costs are a sum of the individual ants' step costs**. What's the minimum state space size for this task, expressed in terms of K and S?

S^K

In this case we need to know the state of each ant, and there are S possibilities for each, hence S^K.

Q1: Ants: Escape!

(iii) Now suppose that each ant i can exit at any of the goal locations G_j, but no two ants can occupy the same square **if they are facing the same direction**. What's the minimum state space size for this task, expressed in terms of K and S?

(iv) Now suppose, once again, that each ant i must reach its own exit at G_i, and no two ants can occupy the same square **if they are facing the same direction**. Let $H = \sum_i h_i^*$, where h_i^* is the optimal cost for ant i to reach goal G_i when it is the only ant in the maze. Is H admissible for the K-ant problem? Select all appropriate answers.

- ☐ Yes, because for any multiagent problem the sum of individual agent costs, with each agent solving a subproblem separately, is always a lower bound on the joint cost.
- ☐ Yes, because H is the exact cost for a relaxed version of the K-ant problem.
- ☐ Yes, because the "no two ants…" condition can only make the true cost larger than H, not smaller.
- ☐ No, because some ants can exit earlier than others so the sum may overestimate the total cost.
- ☐ No, it should be $\max_i$ rather than $\sum_i$.
- ○ None of the above

Q1: Ants: Escape!

(iii) Now suppose that each ant i can exit at any of the goal locations G_j, but no two ants can occupy the same square **if they are facing the same direction**. What's the minimum state space size for this task, expressed in terms of K and S?

$\binom{S}{K}$

For this case, the ants are exchangeable, i.e., you could switch ants i and j and the state would be effectively the same, with the same solution cost. So the state is defined by choosing any K distinct states from the S possibilities.

(iv) Now suppose, once again, that each ant i must reach its own exit at G_i, and no two ants can occupy the same square **if they are facing the same direction**. Let $H = \sum_i h_i^*$, where h_i^* is the optimal cost for ant i to reach goal G_i when it is the only ant in the maze. Is H admissible for the K-ant problem? Select all appropriate answers.

- ☐ Yes, because for any multiagent problem the sum of individual agent costs, with each agent solving a subproblem separately, is always a lower bound on the joint cost.
- ■ Yes, because H is the exact cost for a relaxed version of the K-ant problem.
- ■ Yes, because the "no two ants..." condition can only make the true cost larger than H, not smaller.
- ☐ No, because some ants can exit earlier than others so the sum may overestimate the total cost.
- ☐ No, it should be $\max_i$ rather than $\sum_i$.
- ○ None of the above

This is a relaxed problem, because it is the exact cost for the case where ants are allowed to occupy the same state. The "no two ants" explanation is a less formal but still correct way to argue that H is a lower bound on the true cost.

Q1: Ants: Escape!

(b) The ant is alone again in the maze. Now, the spider will return in T timesteps, so the ant must reach an exit in T or fewer actions. Any sequence with more than T actions doesn't count as a solution.

In this part, we'll address this by solving the original problem and checking the resulting solution. That is, suppose p is a problem and A is a search algorithm; $A(p)$ returns a solution s, and $\ell(s)$ is the length (number of actions) of s, where $\ell(\mathit{failure}) = \infty$. Let p_T be p with the added time limit T. Then, given A, we can define a new algorithm $A'(p_T)$ as follows:

- $s \leftarrow A(p)$; **if** $\ell(s) \leq T$ **then return** s **else return** *failure*.

(i) ◯ T ◯ F Suppose A is an optimal algorithm for p, action costs are 1; then A' is optimal for p_T.

(ii) ◯ T ◯ F Suppose A is a complete algorithm for p; then A' is complete for p_T.

(iii) ◯ T ◯ F Suppose A is an optimal algorithm for p, and action costs may be any nonnegative real number; then A' is optimal for p_T.

(c) Now we attempt to solve the time-limited problem by *modifying the problem definition* (specifically, the states, legal actions in each state, and/or goal test) appropriately so that regular, unmodified search algorithms will automatically avoid returning solutions with more than T actions.

(i) Is this possible *in general, for any problem* where actions costs are all 1? Mark all correct answers.

- ☐ Yes, by augmenting the state space only.
- ☐ Yes, by augmenting the state space and modifying the goal test.
- ☐ Yes, by modifying the goal test only.
- ☐ Yes, by augmenting the state space and modifying the legal actions.
- ☐ Yes, by modifying the legal actions only.
- ◯ No, it's not possible in general.

Q1: Ants: Escape!

(b) The ant is alone again in the maze. Now, the spider will return in T timesteps, so the ant must reach an exit in T or fewer actions. Any sequence with more than T actions doesn't count as a solution.

In this part, we'll address this by solving the original problem and checking the resulting solution. That is, suppose p is a problem and A is a search algorithm; $A(p)$ returns a solution s, and $\ell(s)$ is the length (number of actions) of s, where $\ell(\textit{failure}) = \infty$. Let p_T be p with the added time limit T. Then, given A, we can define a new algorithm $A'(p_T)$ as follows:

- $s \leftarrow A(p)$; **if** $\ell(s) \leq T$ **then return** s **else return** *failure*.

(i) ● T ○ F Suppose A is an optimal algorithm for p, action costs are 1; then A' is optimal for p_T.

For the case where actions costs are 1, an optimal solution is also the shortest in terms of number of actions. If the optimal solution is longer than T, then there is no solution at all.

(ii) ○ T ● F Suppose A is a complete algorithm for p; then A' is complete for p_T.

A complete algorithm need not be optimal (e.g., DFS graph search). Thus A may return a solution longer than T even when a shorter one exists, and thus A' returns failure even when a solution exists for p_T.

(iii) ○ T ● F Suppose A is an optimal algorithm for p, and action costs may be any nonnegative real number; then A' is optimal for p_T.

Here, the least-cost solution for p may have more than T actions if it happens to contain a long sequence of low-cost actions; thus, A' may fail to return the optimal solution for p_T because *failure* is definitely not the optimal solution!

Q1: Ants: Escape!

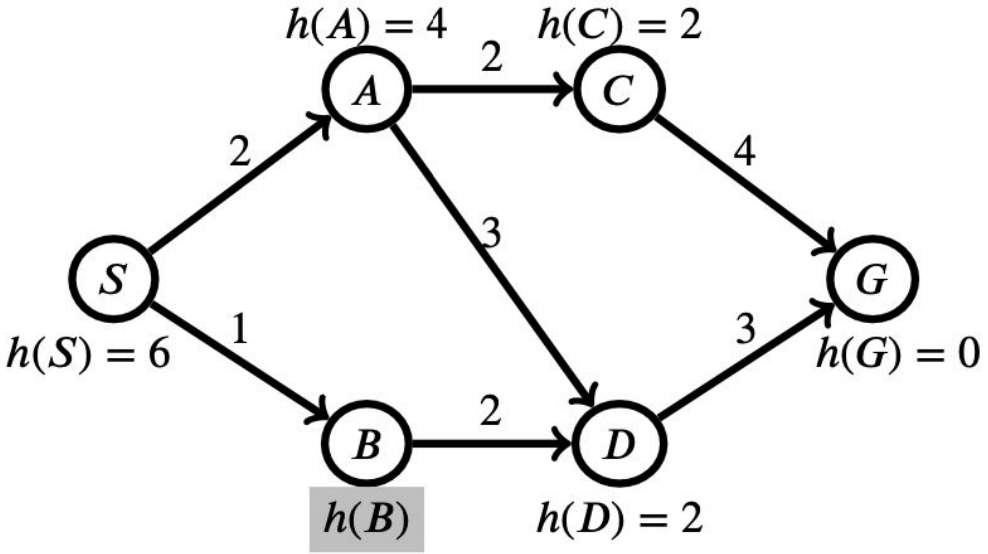
(c) Now we attempt to solve the time-limited problem by *modifying the problem definition* (specifically, the states, legal actions in each state, and/or goal test) appropriately so that regular, unmodified search algorithms will automatically avoid returning solutions with more than T actions.

(i) Is this possible *in general, for any problem* where actions costs are all 1? Mark all correct answers.

- ☐ Yes, by augmenting the state space only.
- ■ Yes, by augmenting the state space and modifying the goal test.
- ☐ Yes, by modifying the goal test only.
- ■ Yes, by augmenting the state space and modifying the legal actions.
- ☐ Yes, by modifying the legal actions only.
- ○ No, it's not possible in general.

You need to augment the state space to include a count of how many actions have been taken along the path. Every action adds one to this count in addition to its other effects. Then you can either modify the goal test to reject any state whose count is more than T, or modify the legal actions to return an empty set for any state whose count is T.

Q2: Informed Search



Search problem graph: S is the start state and G is the goal state.
Tie-break in alphabetical order.

$h(B)$ is unknown and will be determined in the subquestions.

Q2: Informed Search

(a) In this question, refer to the graph above where the optimal path is $S \to B \to D \to G$. For each of the following subparts, you will be asked to write ranges of $h(B)$. You should represent ranges as $\_\_ \leq h(B) \leq \_\_$. *Heuristic values can be any number including* $\pm\infty$. *For responses of* $\pm\infty$, *you can treat the provided inequalities as a strict inequality.* If you believe that there is no possible range, please write "None" in the left-hand box and leave the right box empty.

(i) What is the range for the heuristic to be admissible?

[ ] $\leq h(B) \leq$ [ ]

(ii) What range of heuristic values for B would allow A* tree search to still return the optimal path ($S \to B \to D \to G$)?

[ ] $\leq h(B) \leq$ [ ]

(iii) Now assume that the edges in the graph are undirected (which is equivalent to having two directed edges that point at both directions with the same cost as before). Regardless of whether the heuristic is consistent, what range of heuristic values for B would allow A* tree search to still return the optimal path ($S \to B \to D \to G$)?

[ ] $\leq h(B) \leq$ [ ]

Q2: Informed Search

(a) In this question, refer to the graph above where the optimal path is $S \to B \to D \to G$. For each of the following subparts, you will be asked to write ranges of $h(B)$. You should represent ranges as $\_\_ \leq h(B) \leq \_\_$. *Heuristic values can be any number including $\pm\infty$. For responses of $\pm\infty$, you can treat the provided inequalities as a strict inequality.* If you believe that there is no possible range, please write "None" in the left-hand box and leave the right box empty.

(i) What is the range for the heuristic to be admissible?

0	$\leq h(B) \leq$	5

(ii) What range of heuristic values for B would allow A* tree search to still return the optimal path ($S \to B \to D \to G$)?

$-\infty$	$\leq h(B) \leq$	6

In order to ensure that A* still returns an optimal path, we need to make sure that B gets expanded before D (from A-D). Therefore, $f(D) = 2 + 3 + 2 = 7$ from the A-direction and we need $f(B) = 1 + h(B) \leq 7$ so $h(B) = 6$.

(iii) Now assume that the edges in the graph are undirected (which is equivalent to having two directed edges that point at both directions with the same cost as before). Regardless of whether the heuristic is consistent, what range of heuristic values for B would allow A* tree search to still return the optimal path ($S \to B \to D \to G$)?

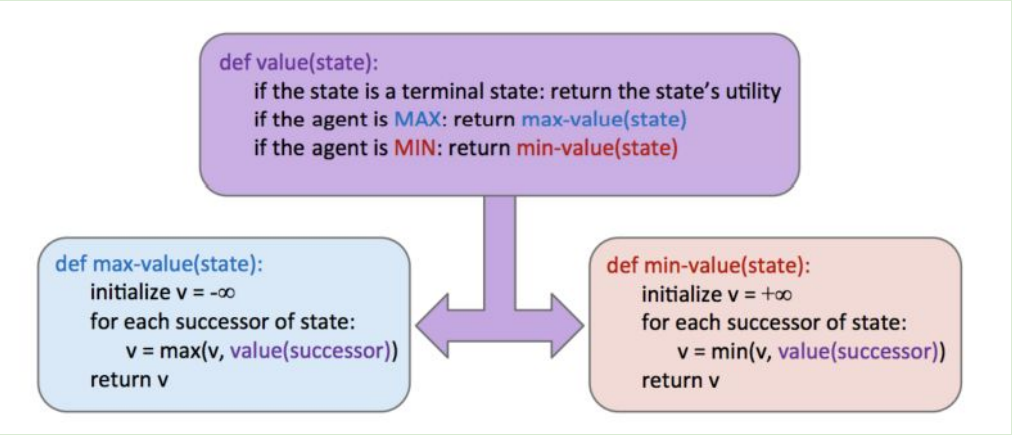
$-\infty$	$\leq h(B) \leq$	6

Solution would still be the same because the heuristic value may cause the algorithm to re-expand nodes B and D but it will still return the same path once the backward cost is greater than the large negative heuristic cost of $h(B)$.

—-- Games --—

Minimax

- Zero-sum game algorithm, where both players play optimally:
  - Maximizer nodes choose actions that maximizes utility
  - Minimizer nodes choose actions that minimize utility
- Agents take turn making moves



$$\forall \text{ agent-controlled states}, \quad V(s) = \max_{s' \in successors(s)} V(s')$$

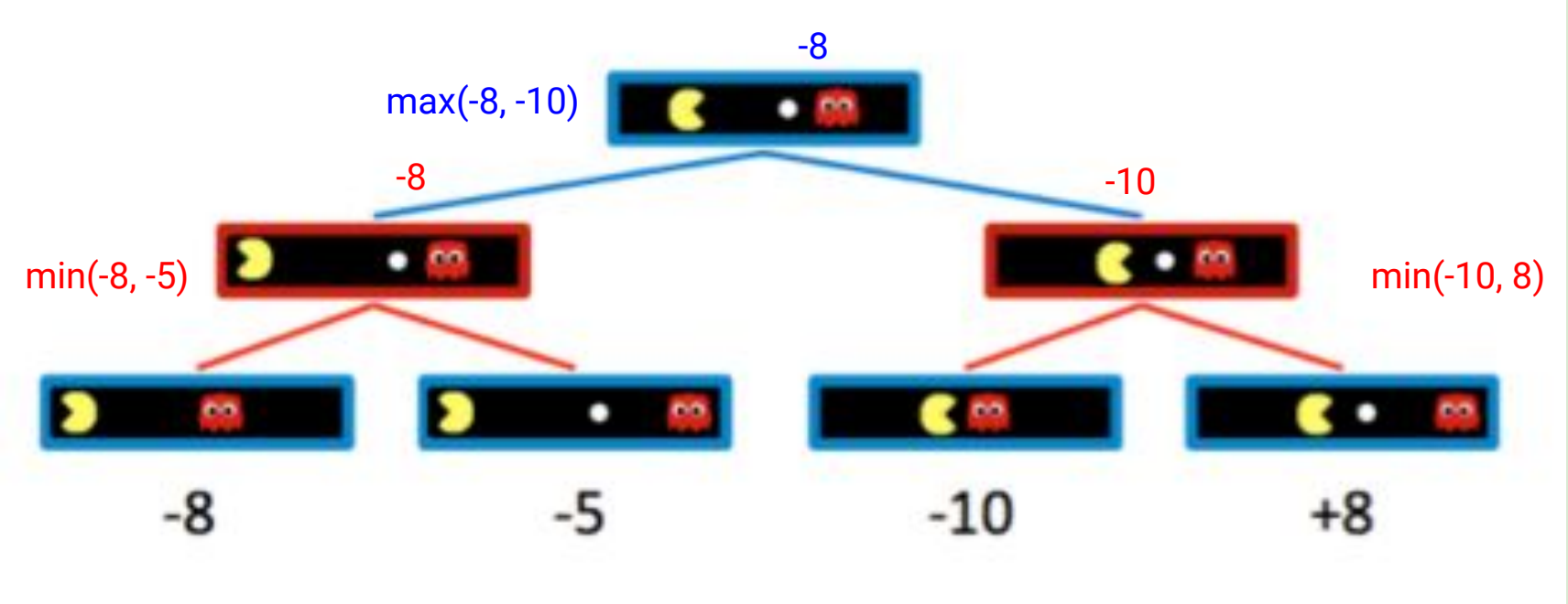
$$\forall \text{ opponent-controlled states}, \quad V(s) = \min_{s' \in successors(s)} V(s')$$

$$\forall \text{ terminal states}, \quad V(s) = \text{known}$$

Minimax Example

$$\forall \text{ agent-controlled states}, \quad V(s) = \max_{s' \in successors(s)} V(s')$$
$$\forall \text{ opponent-controlled states}, \quad V(s) = \min_{s' \in successors(s)} V(s')$$
$$\forall \text{ terminal states}, \quad V(s) = \text{known}$$

Pacman-controlled nodes
Ghost-controlled nodes



Observation

- The value in the maximizer never decreases (and vice versa)
  - We can intelligently use this fact to make multi-agent search more efficient

Alpha Beta Pruning

- Minimax runtime is $O(b^m)$:(
  - b = branching factor, m = approx tree depth where terminal nodes are found
- **Alpha-Beta Pruning:**
  - Say we're looking at some node n. When we go through n's children, if we realize that node n's value is guaranteed to be made redundant by another node, stop checking n's children!
  - Runtime about $O(b^{m/2})$

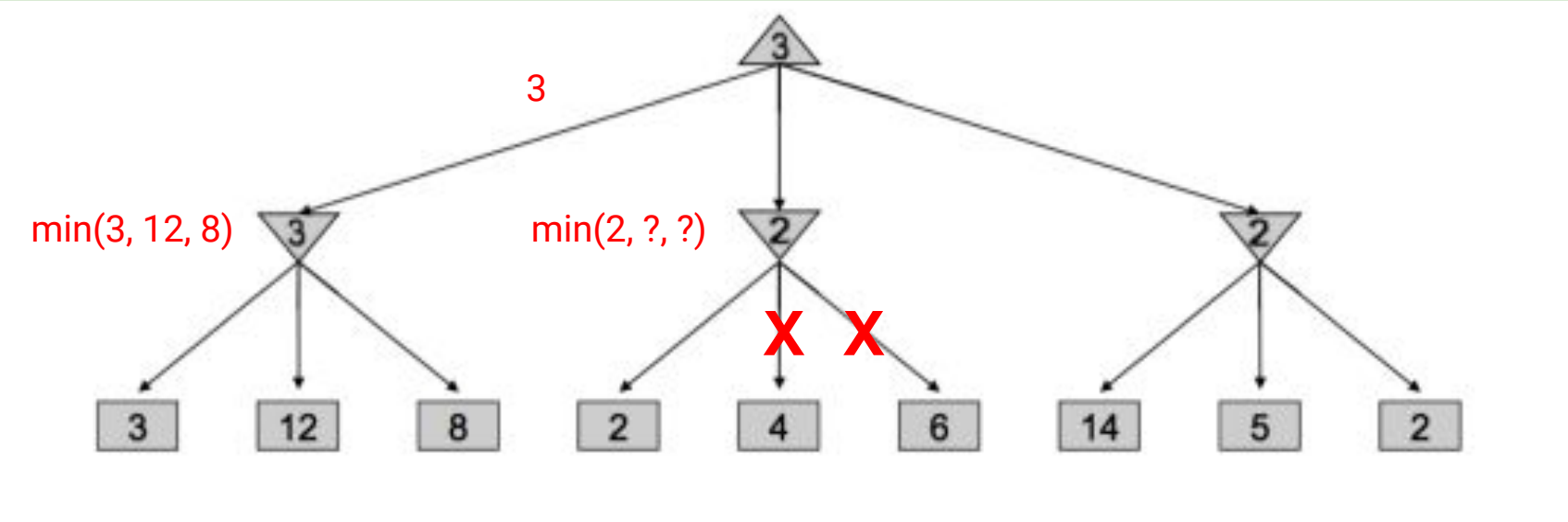
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Example

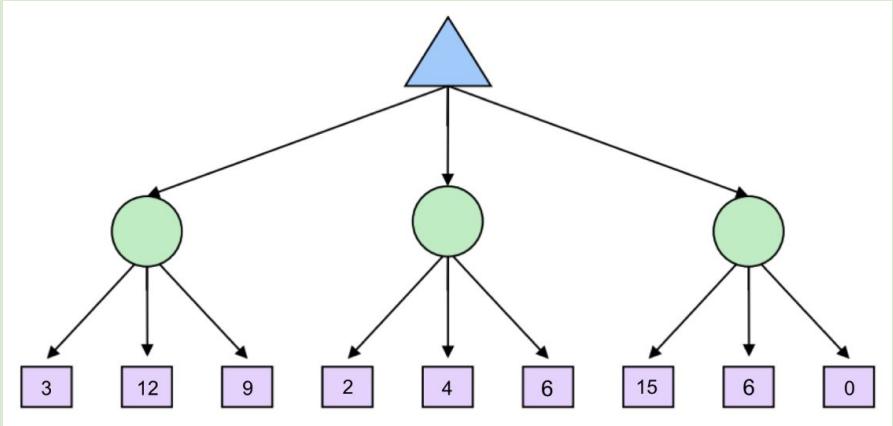
- Maximizer △
- Minimizer ▽



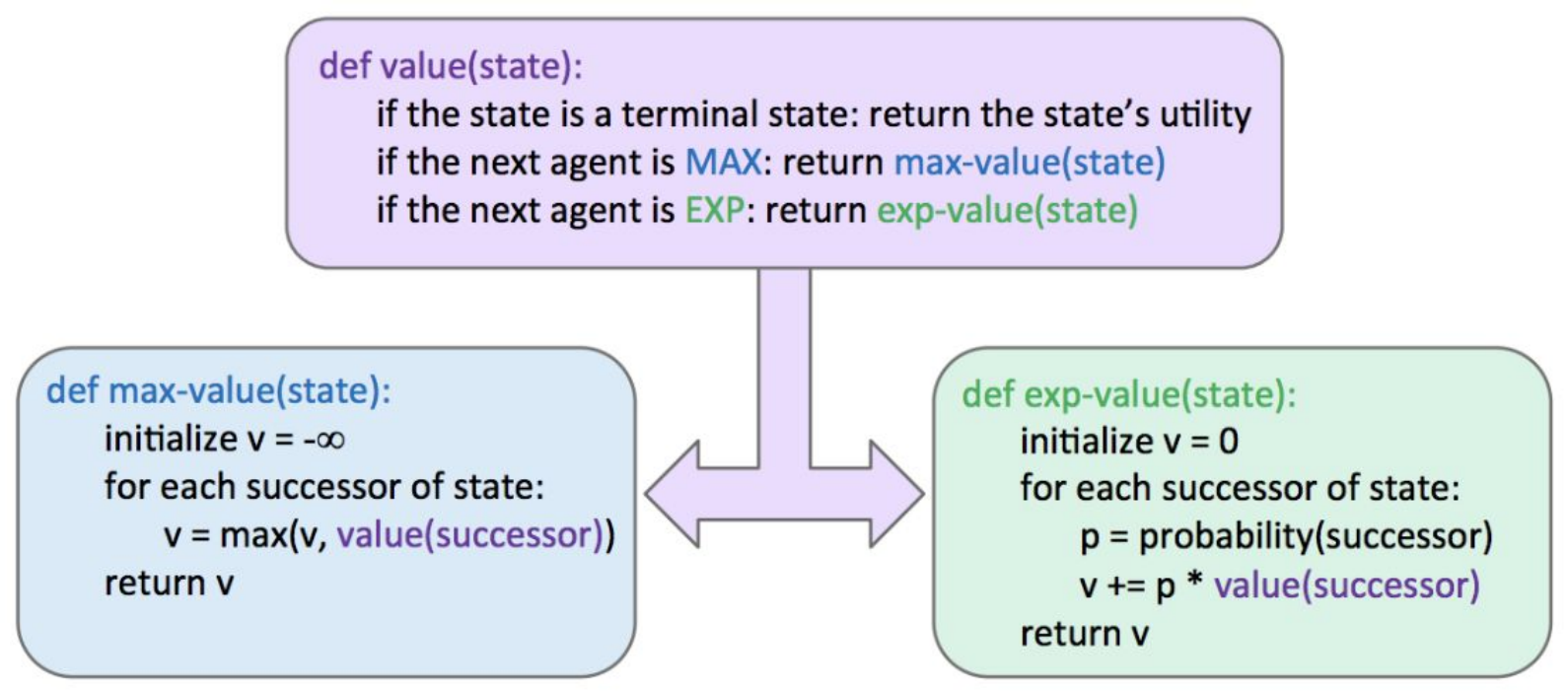
Expectimax

- Adversaries choose actions with some randomness (each action assigned a probability)
- Minimizer nodes replaced by chance nodes, which find the expected value of their children
- Captures non-optimal behavior

$$\begin{aligned} \forall \text{ agent-controlled states}, \quad V(s) &= \max_{s' \in successors(s)} V(s') \\ \forall \text{ chance states}, \quad V(s) &= \sum_{s' \in successors(s)} p(s'|s) V(s') \\ \forall \text{ terminal states}, \quad V(s) &= \text{known} \end{aligned}$$



Expectimax Pseudocode

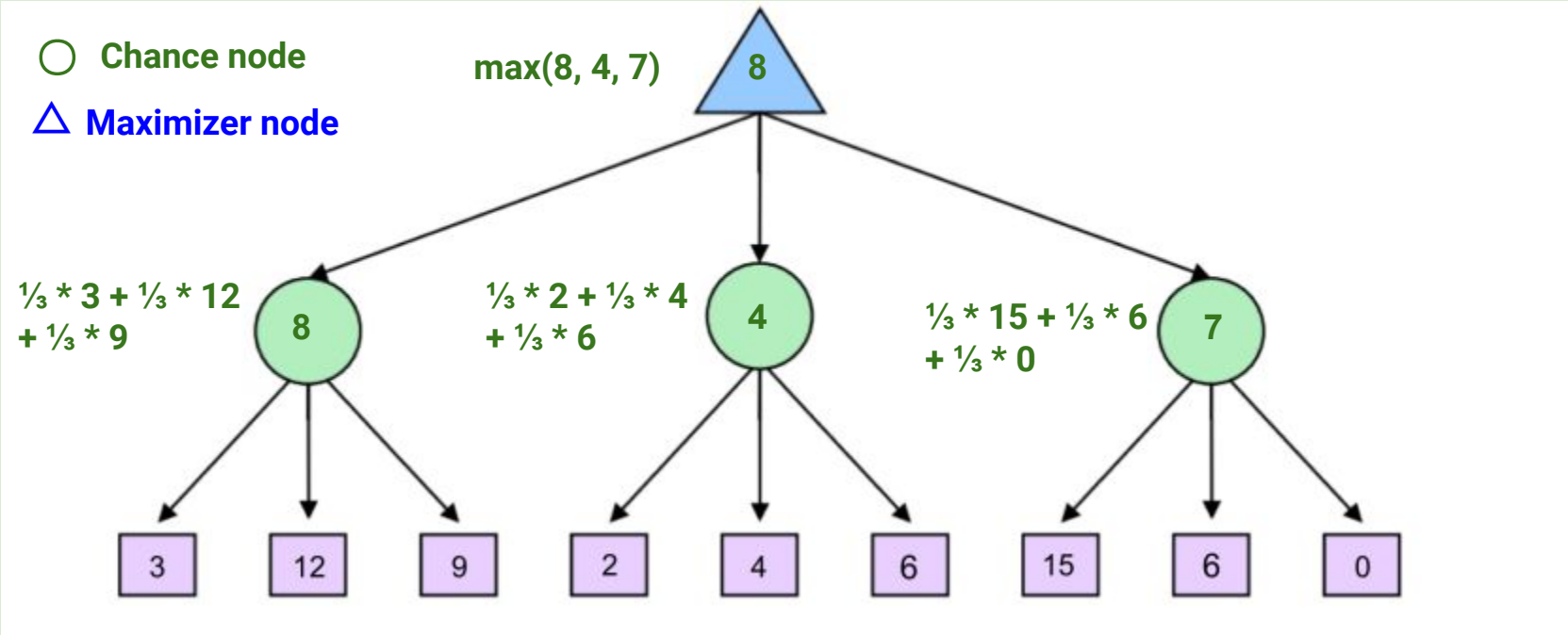


Expectimax Example

$$\forall \text{agent-controlled states}, \quad V(s) = \max_{s' \in successors(s)} V(s')$$

$$\forall \text{chance states}, \quad V(s) = \sum_{s' \in successors(s)} p(s'|s) V(s')$$

$$\forall \text{terminal states}, \quad V(s) = \text{known}$$

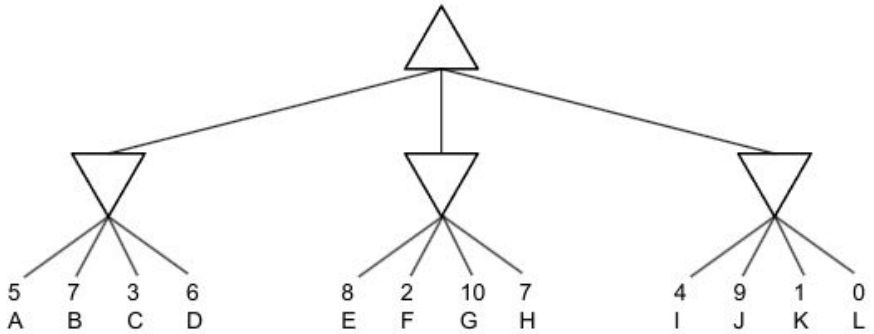


Games

(a) Minimax and Alpha-Beta Pruning We have a two-player, zero-sum game with k rounds. In each round, the maximizer acts first and chooses from n possible actions, then the minimizer acts next and chooses from m possible actions. After the minimizer's k-th turn, the game finishes and we arrive at a utility value (leaf node). Both players behave optimally. Explore nodes from left to right.

(i) What is the total number of leaf nodes in the game tree, in terms of m, n, k? ☐

(ii) In the minimax tree below $k = 1, n = 3, m = 4$.



(1) What is the minimax value of the root? ☐

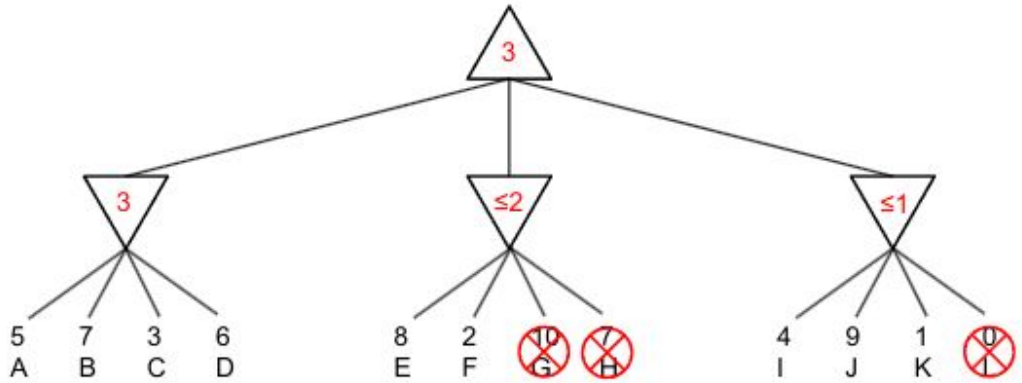
(2) Which leaf nodes are pruned by alpha-beta? Mark the corresponding letters below.

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K ☐ L ○ None

(a) Minimax and Alpha-Beta Pruning We have a two-player, zero-sum game with k rounds. In each round, the maximizer acts first and chooses from n possible actions, then the minimizer acts next and chooses from m possible actions. After the minimizer's k-th turn, the game finishes and we arrive at a utility value (leaf node). Both players behave optimally. Explore nodes from left to right.

(i) What is the total number of leaf nodes in the game tree, in terms of m, n, k? $(mn)^k$

(ii) In the minimax tree below $k = 1, n = 3, m = 4$.



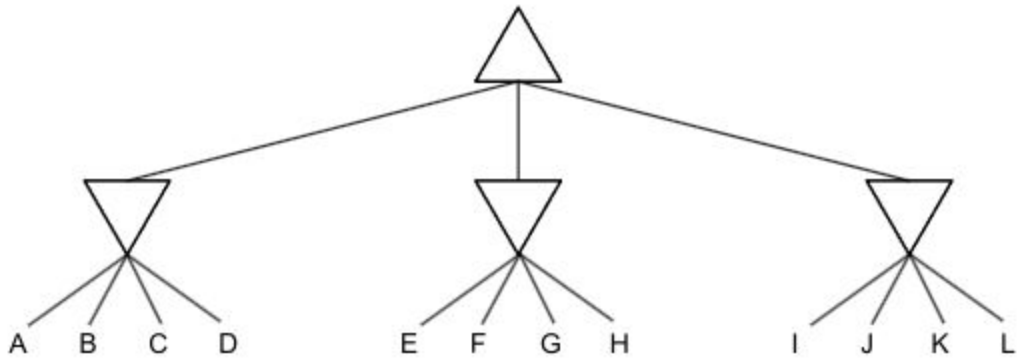
(1) What is the minimax value of the root? 3

(2) Which leaf nodes are pruned by alpha-beta? Mark the corresponding letters below.

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ■ G ■ H ☐ I ☐ J ☐ K ■ L ○ None

(iii) When $k = 1$, in the *best case* (pruning the most nodes possible), which nodes can be pruned in the tree below?

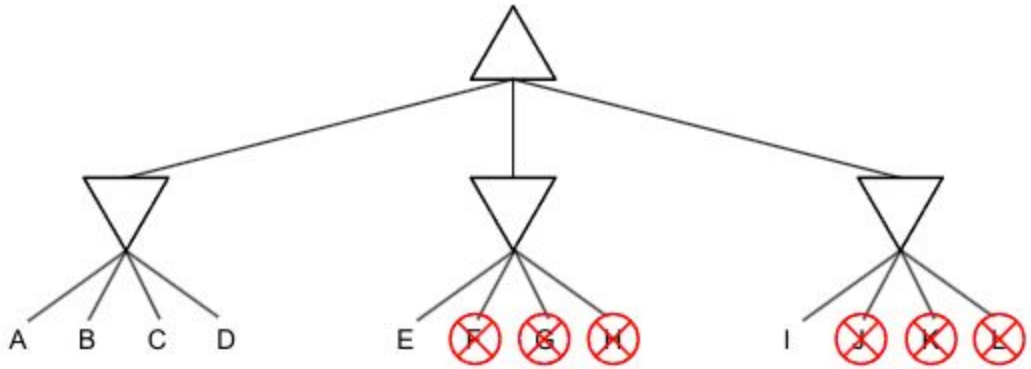
☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K ☐ L ○ None



(iv) Now consider the same $k = 1$ but with a general m and n number of maximizer and minimizer actions respectively. How many leaf nodes would be pruned in the best case? Express your answer in terms of m and n.

(iii) When $k = 1$, in the *best case* (pruning the most nodes possible), which nodes can be pruned in the tree below?

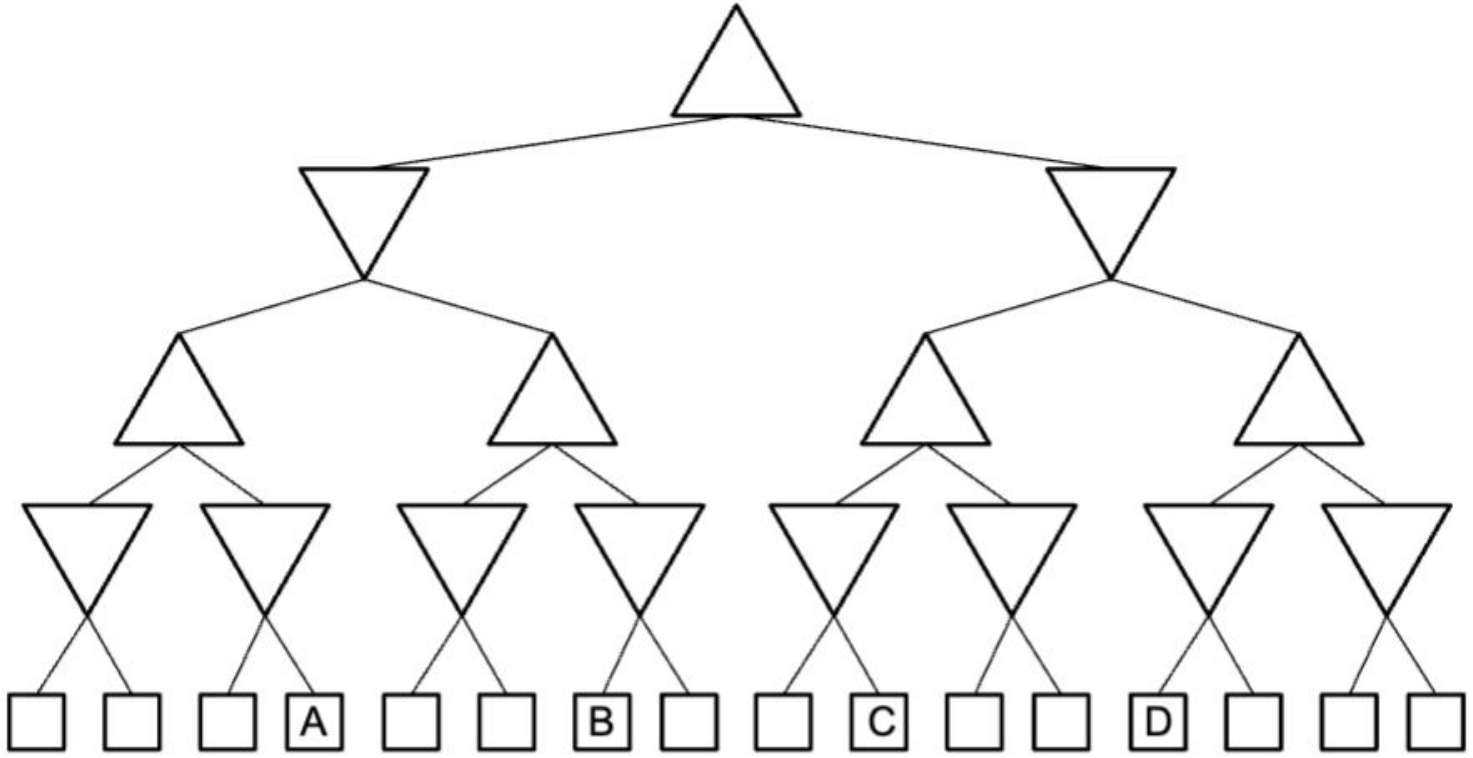
☐ A ☐ B ☐ C ☐ D ☐ E ■ F ■ G ■ H ☐ I ■ J ■ K ■ L ○ None



(iv) Now consider the same $k = 1$ but with a general m and n number of maximizer and minimizer actions respectively. How many leaf nodes would be pruned in the best case? Express your answer in terms of m and n.

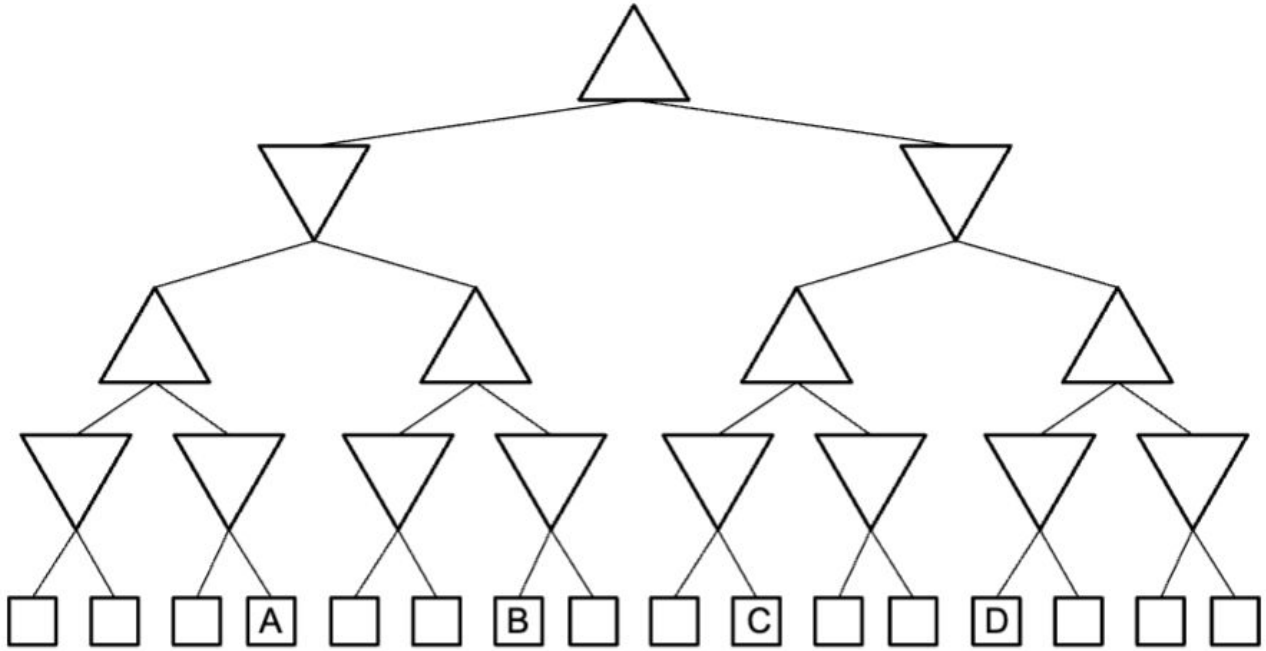
$(n-1)(m-1)$

(v) When $k = 2, n = 2, m = 2$, in the best case, which of the leaves labeled A, B, C, D will be pruned?



☐ A ☐ B ☐ C ☐ D ○ None

(v) When $k = 2, n = 2, m = 2$, in the best case, which of the leaves labeled A, B, C, D will be pruned?



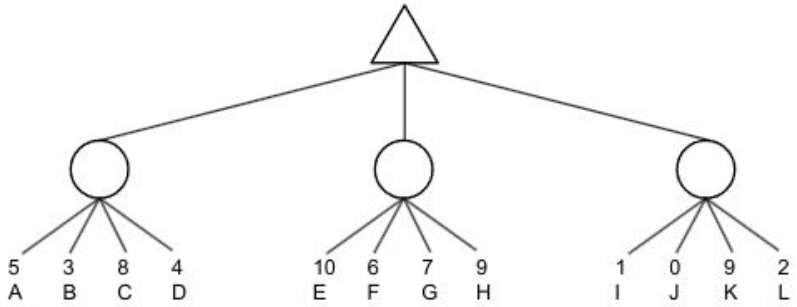
■ A ■ B ■ C ■ D ○ None

The key to identifying best-case pruning is to recognize that when looking at a node with p branches, we can prune the $(p - 1)$ right branches if we are coming into that node with a finite alpha or beta value. In order to have a finite alpha (or beta) value, the maximizer (or minimizer) must already have a better option on the path to the root. This is only possible if we've already seen another maximizer (or minimizer) node during our traversal.

(b) Chance Nodes Our maximizer agent is now playing against a non-optimal opponent. In each round, the maximizer acts first, then the opponent acts next and chooses uniformly at random from m possible actions.

(i) Subpart removed for time (simple expectimax)

(ii) Consider the game tree below where we now **know** that the opponent always has $m = 4$ possible moves and chooses uniformly at random. **We also know that all leaf node utility values are less than or equal to $c = 10$.**



(1) What is the value of the root node? ☐

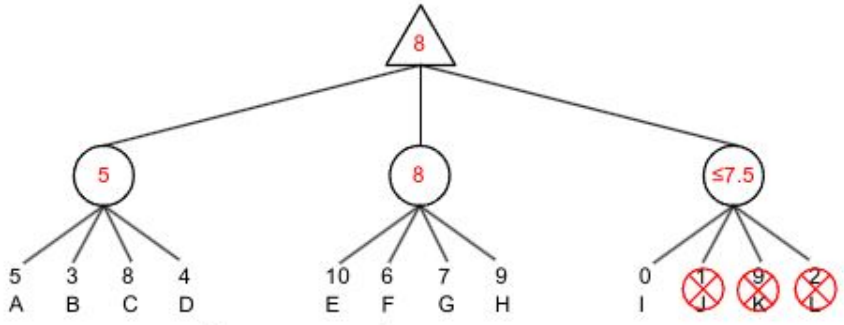
(2) Which nodes can be pruned?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K ☐ L ○ None

(c) Now, let's generalize this idea for pruning on expectimax. We consider expectimax game trees where the opponent always chooses uniformly at random from m possible moves, and all leaf nodes have values no more than c. These facts are known by the maximizer player.

(i) Let's say that our depth-first traversal of this game tree is currently at a chance node and has seen k children of this node so far. The sum of the children seen so far is S. What is the largest possible value that this chance node can take on? (Answer in terms of m, c, k, and S)

☐



(1) What is the value of the root node? 8

(2) Which nodes can be pruned?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ■ J ■ K ■ L ○ None

(c) Now, let's generalize this idea for pruning on expectimax. We consider expectimax game trees where the opponent always chooses uniformly at random from m possible moves, and all leaf nodes have values no more than c. These facts are known by the maximizer player.

(i) Let's say that our depth-first traversal of this game tree is currently at a chance node and has seen k children of this node so far. The sum of the children seen so far is S. What is the largest possible value that this chance node can take on? (Answer in terms of m, c, k, and S)

$\frac{1}{m}(S + (m-k)c)$

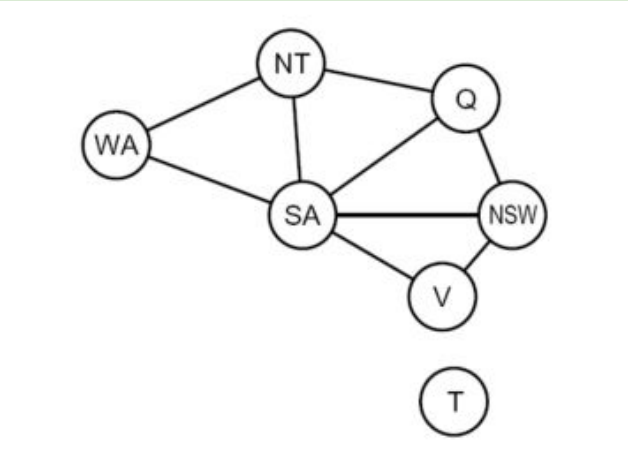
—-- CSPs --—

Constraint Satisfaction Problems

- **CSPs:**
  - Special subset of search problems
  - State defined by variables $\mathbf{X_1, \ldots, X_n}$ from domain **D** (sometimes $D_1, \ldots, D_n$)
  - Goal test is set of constraints on variables
- **Constraints:**
  - **Unary:** a single variable in the CSP, used to prune domain of the variable
  - **Binary:** involve 2 variables, represented as line on CSP graph
  - **Higher-order:** involve 3+ variables
  - **Implicit vs Explicit:** explicit constraints explicitly list out possible variable arrangements, we'll see an example on the next slide

CSP Example: Map Coloring

- Constraint: No adjacent states can be the same color
- Implicit: WA ≠ NT (bordering edges)
- Explicit: (WA, NT) ∈ {(red, green), (red, blue), ...}

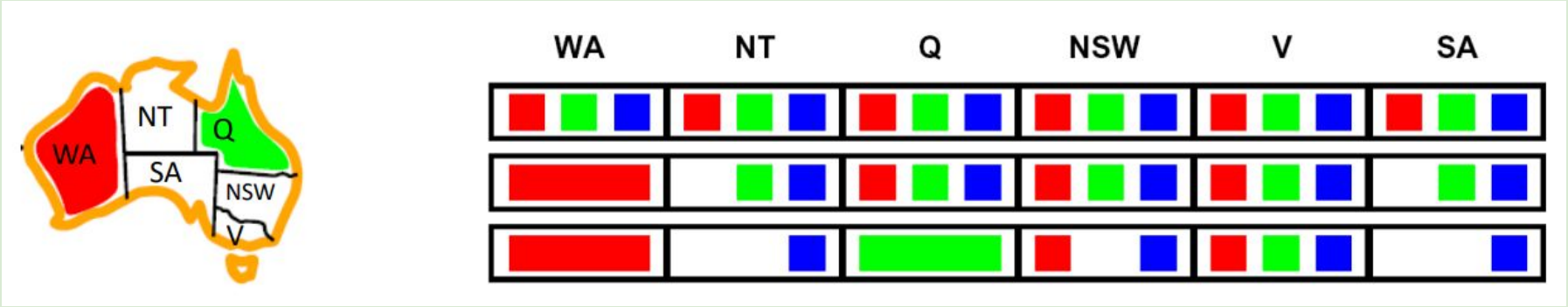


Backtracking Search

- Optimization on DFS
- Fix ordering of variables & select values in this order
- When selecting new value for variable, don’t conflict with previous assignments!
    - If this isn’t possible, then **backtrack** - return to previous variable and change its value

Filtering: Forward Checking

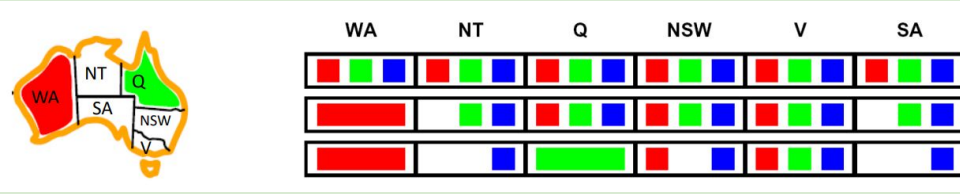
- **Filtering:** Keep track of domains for unassigned variables and cross off bad options
- **Forward checking:** Cross off values that violate a constraint when added to the existing assignment
    - Run after every variable assignment



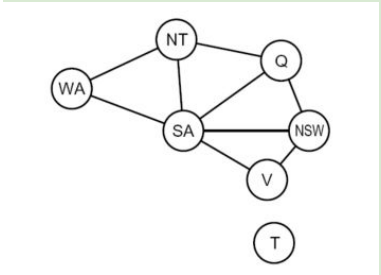
Ordering(Or Heuristics)

- **Heuristics:**
  - Allow us to choose assignments by how likely they are to lead to a solution
- **Minimum Remaining Values (MRV):**
  - Choose variable to assign next with least remaining possible values in its domain
- **Least Constraining Value (LCV):**
  - Given a variable, choose the least constraining value to assign to it
  - Choose value for variable that rules out the fewest options for other variables

Summary



- **CSPs:**
  - Special subset of search problems
  - State defined by variables $\mathbf{X_1}$, …, $\mathbf{X_n}$ from domain **D** (sometimes D_1, …, D_n)
  - Goal test is set of constraints on variables
- **Implicit vs Explicit:** explicit constraints explicitly list out possible variable arrangements
- **Backtracking Search:**
  - Assign variables in order, preserving constraints, until not possible - then backtrack

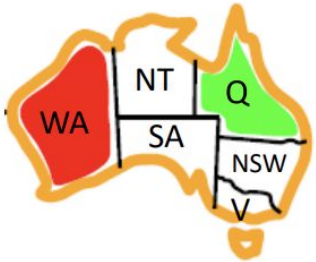


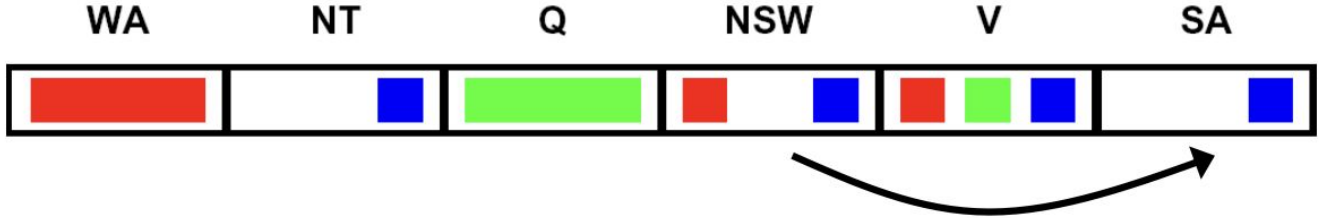
- **Forward Checking**
  - Cross off values that violate a constraint when added to the existing assignment
  - Run after each variable assignment

- **Minimum Remaining Values (MRV):**
  - Choose variable to assign next with least remaining possible values in its domain
- **Least Constraining Value (LCV):**
  - Given a variable, choose the least constraining value to assign to it
  - Choose value for variable that rules out the fewest options for other variables

Arc Consistency

- An **arc** is really just a pair of variables with an associated direction
  (and a “head” [head of the arrow] and “tail”)
- An arc is **consistent** if for every value in the tail, the head is not impossible
- This arc is **inconsistent** because if we assigned NSW to blue, SA would be impossible
  (all options violate some constraint)

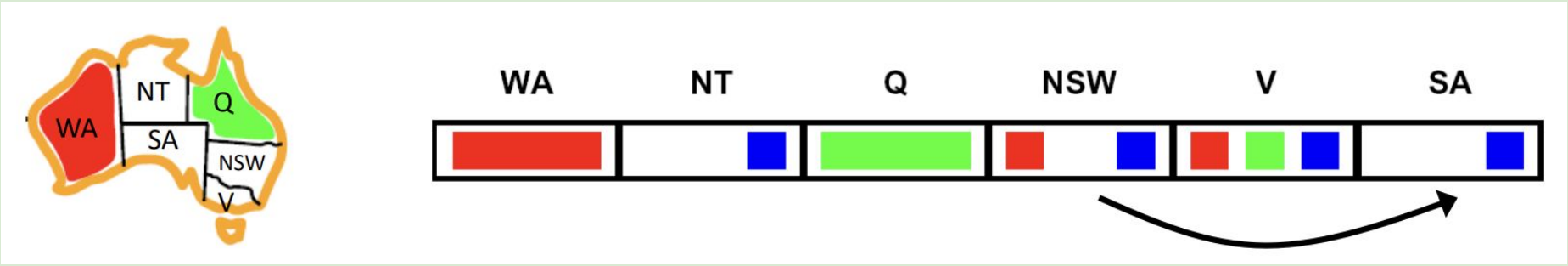




- We can remove values from the tail that lead to inconsistency

Filtering: Arc Consistency

- We can filter using arc consistency by trying to ensure all arcs are consistent (which means deleting values that would lead to certain failure!)



CSP

You are a newly appointed zookeeper, and your first task is to find rooms for all of the animals.

The zoo has three animals: the Iguana (I), Jaguar (J), and Koala (K).

Each animal needs to be assigned to one of four rooms: the North room (N), East room (E), South room (S), or West room (W), subject to the following constraints:

1. The jaguar cannot share a room with any other animal.
2. The iguana and koala must be in different rooms.
3. The koala can only be in the East room or the South room.

(a) Consider the first constraint: "The jaguar cannot share a room with any other animal."

Can this constraint be expressed using only binary constraints on the three variables I, J, K?

- ◯ Yes, it can be expressed as 1 binary constraint.
- ◯ Yes, it can be expressed as 2 different binary constraints.
- ◯ Yes, it can be expressed as 4 different binary constraints.
- ◯ No, this is necessarily a unary constraint.
- ◯ No, this is necessarily a higher-order constraint.

(a) Consider the first constraint: "The jaguar cannot share a room with any other animal."

Can this constraint be expressed using only binary constraints on the three variables I, J, K?

- ○ Yes, it can be expressed as 1 binary constraint.
- ● Yes, it can be expressed as 2 different binary constraints.
- ○ Yes, it can be expressed as 4 different binary constraints.
- ○ No, this is necessarily a unary constraint.
- ○ No, this is necessarily a higher-order constraint.

Recall that a binary constraint relates two variables (in this problem, two animals). We can write this constraint as two different binary constraints: The jaguar cannot share a room with the iguana. The jaguar cannot share a room with the koala. $J \neq I$ and $J \neq K$.

(b) Suppose we enforce unary constraints, and then assign the jaguar to the South room. The remaining values in each domain would be:

Iguana:	North, East, South, West
Jaguar:	South
Koala:	East, South

In the table below, mark each value that would be **removed** by running forward-checking after this assignment.

	North	East	South	West
Iguana	☐	☐	☐	☐
Koala		☐	☐	

(c) Regardless of your answer to the previous subpart, suppose we've done some filtering and the following values remain in each variable's domain:

Iguana:	East, West
Jaguar:	South
Koala:	East

Are all the arcs in this CSP consistent?

- ◯ Yes, all arcs are consistent.
- ◯ No, only $K \rightarrow I$ is not consistent.
- ◯ No, only $I \rightarrow K$ is not consistent.
- ◯ No, $K \rightarrow I$ and $I \rightarrow K$ are both not consistent.
- ◯ No, only $J \rightarrow I$ is not consistent.
- ◯ No, only $I \rightarrow J$ is not consistent.
- ◯ No, $J \rightarrow I$ and $I \rightarrow J$ are both not consistent.

(b) Suppose we enforce unary constraints, and then assign the jaguar to the South room. The remaining values in each domain would be:

Iguana:	North, East, South, West
Jaguar:	South
Koala:	East, South

In the table below, mark each value that would be **removed** by running forward-checking after this assignment.

	North	East	South	West
Iguana	☐	☐	■	☐
Koala		☐	■	

The main constraint relating the jaguar to the other variables is the first constraint. Since the jaguar has been assigned to the South room, no other animal can be assigned to the South room. Forward checking does not propagate any further.

Formally, forward checking considers every other variable in relation to the variable that was just assigned (here, that's J).

Are all the arcs in this CSP consistent?

- ○ Yes, all arcs are consistent.
- ○ No, only $K \rightarrow I$ is not consistent.
- ● No, only $I \rightarrow K$ is not consistent.
- ○ No, $K \rightarrow I$ and $I \rightarrow K$ are both not consistent.
- ○ No, only $J \rightarrow I$ is not consistent.
- ○ No, only $I \rightarrow J$ is not consistent.
- ○ No, $J \rightarrow I$ and $I \rightarrow J$ are both not consistent.

The $I \rightarrow K$ arc is not consistent. If we assign Iguana to East, there is no valid assignment to Koala that would satisfy the constraints (because of constraint 2).

All of the other arcs in this CSP are consistent. We can go through and check the arcs listed as options in this subpart:

$K \rightarrow I$ is consistent because if we assign East at K, we can still assign West at I.

$J \rightarrow I$ is consistent because if we assign South at J, we can still assign East or West at I.

$I \rightarrow J$ is still consistent because if we assign East at I, South is a valid assignment at J. Also, if we assign West at I, South is a valid assignment at J.

For completeness, here are the remaining arcs:

$J \rightarrow K$ is consistent. If we assign South at J, it's fine to assign East at K.

$K \rightarrow J$ is consistent. If we assign East at K, it's fine to assign South at J.

The constraints, repeated here for your convenience:

1. The jaguar cannot share a room with any other animal.
2. The iguana and koala must be in different rooms.
3. The koala can only be in the East room or the South room.

(d) Regardless of your answer to the previous subpart, suppose we start over and just enforce the third constraint. Then the remaining values in each domain are:

Iguana:	North, East, South, West
Jaguar:	North, East, South, West
Koala:	East, South

What does the minimum remaining values (MRV) heuristic suggest doing next?

- ◯ Assign North or West to a variable next.
- ◯ Assign East or South to a variable next.
- ◯ Assign a value to Koala next.
- ◯ Assign a value to Iguana or Jaguar next.

(e) Again, consider the CSP after just enforcing the third constraint:

Iguana:	North, East, South, West
Jaguar:	North, East, South, West
Koala:	East, South

Which assignment would the least constraining value (LCV) heuristic prefer?

- ◯ Assign North to Jaguar.
- ◯ Assign East to Jaguar.
- ◯ LCV is indifferent between these two assignments.

(d) Regardless of your answer to the previous subpart, suppose we start over and just enforce the third constraint. Then the remaining values in each domain are:

Iguana:	North, East, South, West
Jaguar:	North, East, South, West
Koala:	East, South

What does the minimum remaining values (MRV) heuristic suggest doing next?

- ○ Assign North or West to a variable next.
- ○ Assign East or South to a variable next.
- ● Assign a value to Koala next.
- ○ Assign a value to Iguana or Jaguar next.

MRV tells us which variable to assign next. In particular, MRV suggests assigning the variable with the fewest remaining values in its domain, which here is Koala (with just 2 values left in its domain, compared to 4 values left in the domains of the other variables).

MRV does not tell us which value to assign next, so we can rule out options 1 and 2 (which suggest values to assign).

(e) Again, consider the CSP after just enforcing the third constraint:

Iguana:	North, East, South, West
Jaguar:	North, East, South, West
Koala:	East, South

Which assignment would the least constraining value (LCV) heuristic prefer?

- ● Assign North to Jaguar.
- ○ Assign East to Jaguar.
- ○ LCV is indifferent between these two assignments.

LCV suggests assigning the value that deletes the fewest values in the domains of other variables.

Assuming we use forward-checking to filter, assigning North to Jaguar causes Iguana to lose North as an option. Assigning East to Jaguar causes both Iguana and Koala to lose East as an option.

Assigning North to Jaguar deletes the fewest values, so LCV gives this assignment higher priority.

Intuitively, North is a less constraining option than East for J, because North leaves K with two options, while East only leaves K with one option. Both North and East for J leave I with three options.

(f) Suppose we add another constraint:

"The West room can contain at most one animal."

Can this constraint be expressed using only binary constraints on the three variables I, J, K?

- ◯ Yes, it can be expressed as 1 binary constraint.
- ◯ Yes, it can be expressed as 2 different binary constraints.
- ◯ Yes, it can be expressed as 3 different binary constraints.
- ◯ No, this is necessarily a higher-order constraint.

(f) Suppose we add another constraint:

"The West room can contain at most one animal."

Can this constraint be expressed using only binary constraints on the three variables I, J, K?

- ○ Yes, it can be expressed as 1 binary constraint.
- ○ Yes, it can be expressed as 2 different binary constraints.
- ● Yes, it can be expressed as 3 different binary constraints.
- ○ No, this is necessarily a higher-order constraint.

Recall that a binary constraint relates two variables (in this problem, two animals). We can write this constraint as three different binary constraints: $\neg(J = w \wedge I = w)$ and $\neg(J = w \wedge K = w)$ and $\neg(K = w \wedge I = w)$.

In other words, we go through all pairs of variables, and enforce that no pair of variables can both be assigned to West. There are 3 possible pairs of variables that need to be considered in this problem.

"The iguana and jaguar cannot both be in the West room."

"The jaguar and koala cannot both be in the West room."

"The iguana and koala cannot both be in the West room."