General Problem Solving Approaches & Games

Mahit Kumar Paul
Assistant Professor, Dept. of CSE
RUET, Rajshahi-6204
mahit.cse@gmail.com

Problem and Goal Formulation

Problem formulation is the process of deciding what **actions** and **states** to consider, given a goal.

Example: Visiting Dhaka from Rajshahi

Goal formulation is the first step in problem solving that is based on the current situation and the agent's performance measure.

Well-defined Problems & Solutions

A **problem** can be defined formally by **five components**:

- Initial state where the agent starts in. For example, the initial state for our agent can be described as In(Rajshahi).

- Actions A description of the possible activities available to the agent at a particular state s. For example, from the state In(Rajshahi), the actions are {Go(Food-village), Go(Tangail), Go(Dhaka)}

Well-defined Problems & Solutions…

- Transition Model A description of what each action does specified by a function **RESULT(s, a)** that returns the state that results from doing action **a** in state **s**. For example, we have

  RESULT(In(Rajshahi), Go(Food- village)) = In(Food-village)

- Goal test, which determines whether a given state is a goal state.

- Path cost A function that assigns a numeric cost to each path.

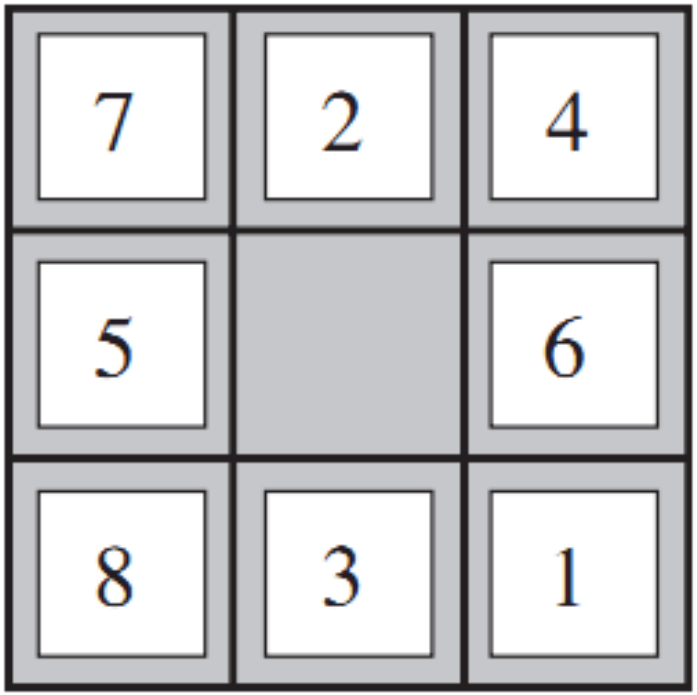
Well-defined Problems & Solutions…

- **Solution** to a problem is an action of sequence that leads from the initial state to a goal state.

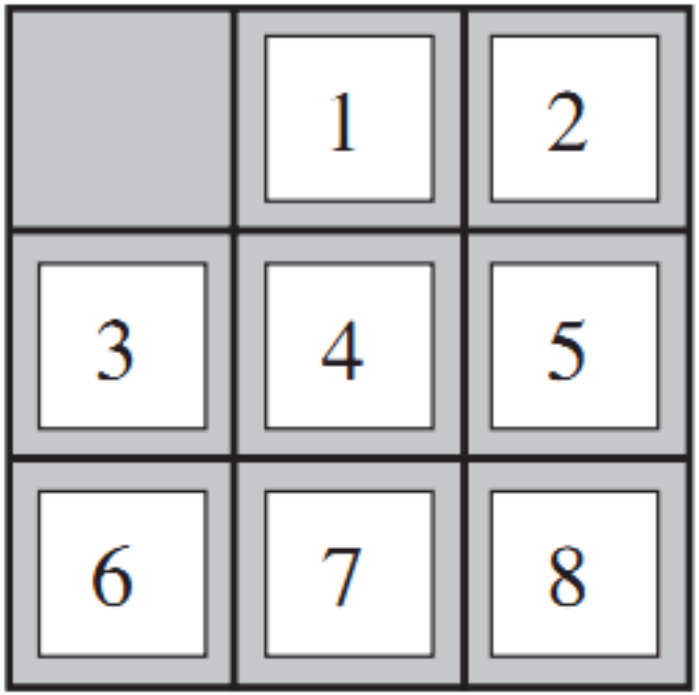
- Solution quality is measured by-

  - ✓ The path cost function
  - ✓ An optimal solution that has the lowest path cost among all solutions.

Example Problem: 8-Puzzle



Start State



Goal State

Example Problem: 8-Puzzle…

- Initial state: Any state can be designated as the initial state.
- Actions: The simplest formulation defines the actions as movements of the blank space Left, Right , Up or Down.
- Transition model: Given a state and action, this returns the resulting state; for example, if we apply Left to the start state the resulting state has the 5 and the blank switched.
- Goal test: This checks whether the state matches the goal.
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

Search Terminology

- State/Problem Space: The initial state, actions, and transition model implicitly define the state space of the problem—the set of all states reachable from the initial state by any sequence of actions.

- Problem Space Graph: Represents problem space. States are shown by nodes and operators i.e. actions are shown by edges.

- Path: A path in the state space is a sequence of states connected by a sequence of actions.

- Step Cost: Cost of taking action a in state s to reach state s' is denoted by **c(s, a, s')**.

Search Terminology…

- Depth of Problem: Shortest sequence of operators from initial state to goal state i.e. **length of a shortest path**.

- Admissibility: A property of an algorithm to always find an optimal solution.

Searching Algorithms

- Searching is the universal technique for problem solving in AI.

- There are some single-player games such as tile games, sudoku, crossword, etc.

- Search algorithms help to search for a particular position in such games.

Single Agent Path-Finding Problems

- Games such as 3x3 eight-tile, 4x4 fifteen tile puzzles are single agent path finding challenges.
- They consist of a matrix of tiles with a blank tile.
- The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objectives.
- Other example of single agent path finding are: TSP (Travelling Salesman Problem), **Rubik's Cube** etc.

Measuring problem-solving performance

Four Ways:

- Completeness: Is the algorithm guaranteed to find a solution when there is one?
- Optimality: Does the strategy find the optimal solution?
- Time Complexity: How long does it take to find a solution?
- Space Complexity: How much memory is needed to perform the search?

Uninformed / Blind Search & Informed / Heuristic Search

- **Uninformed / Blind Search** means that the strategies have no additional information about states beyond that provided in the problem definition.
  - ✓ All they can do is generate successors and distinguish a goal state from a non-goal state.
  - ✓ All search strategies are distinguished by the order in which nodes are expanded.

- **Informed / Heuristic Search** Strategies that know whether one non-goal state is "more promising" than another.

Uninformed Search Strategies

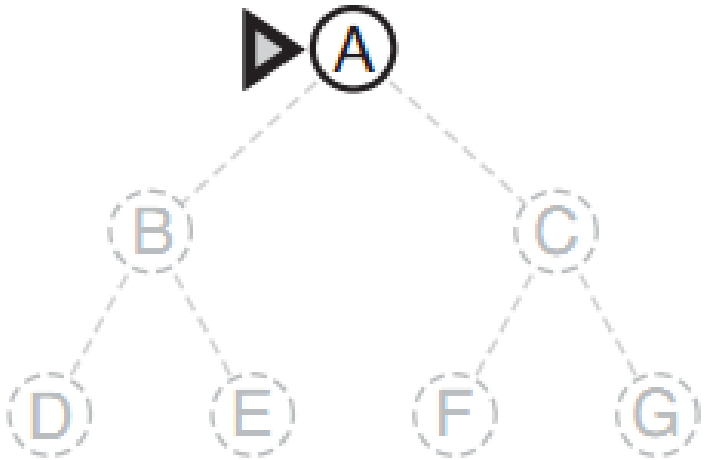
- Breadth First Search (BFS)
- Uniform Cost Search (UCS)
- Depth First Search (DFS)
- Depth Limited Search (DLS)
- Iterative Deepening Search (IDS)

Breadth First Search (BFS)

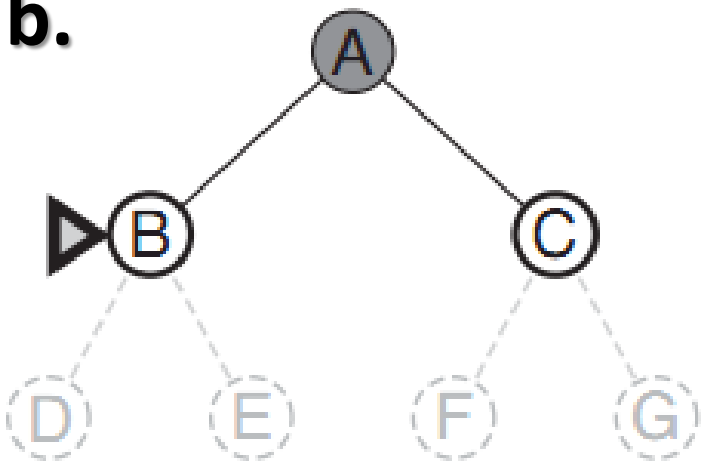
- **Breadth-first Search** is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- **In general**, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- The **Goal Test** is applied to each node when it is generated rather than when it is selected for expansion.

Breadth First Search (BFS)…

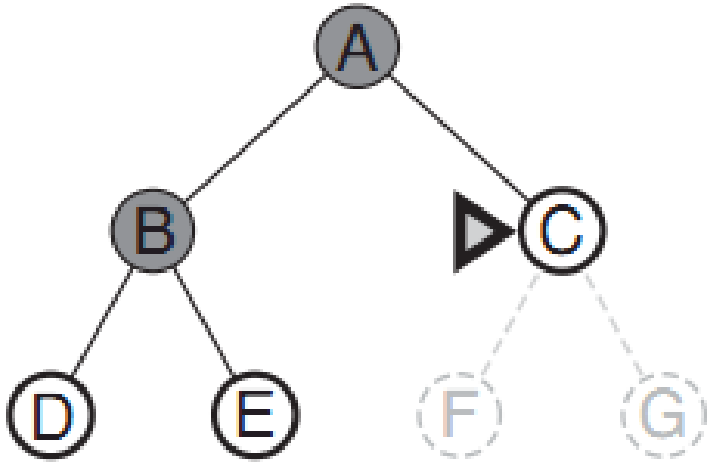
a.



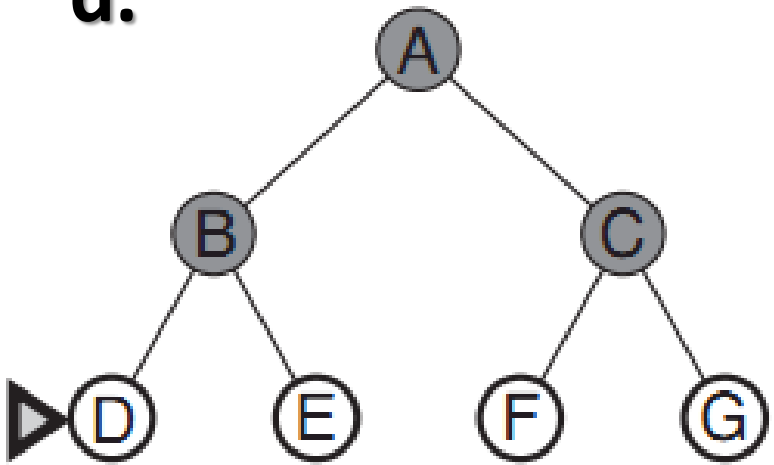
b.



c.



d.

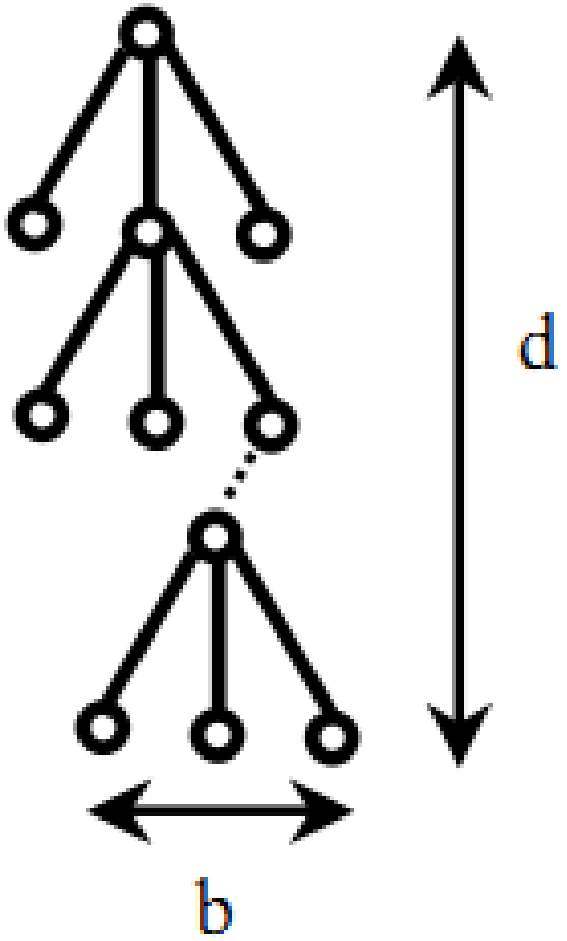


Performance of BFS

1. Completeness – Yes, it always reaches goal (if b and d are finite)

2. Optimality - if the path cost is a non-decreasing function of the depth of the node. i.e. (if we guarantee that deeper solutions are less optimal, e.g. step-cost is Constant (say 1)

Performance of BFS…



Performance of BFS…

3. Space complexity - Suppose that the solution is at depth d. In the worst case, it is the last node generated at that level. Then the total number of nodes generated is **O(b^d)**.

Performance of BFS…

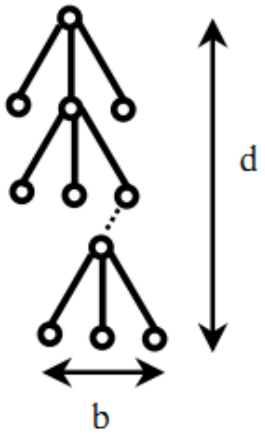
If each node has b immediate descendants, then level 0 (the root node) has 1 node.
Level 1 has b nodes
Level 2 has $b*b = b^2$ nodes
Level 3 has $b^2*b = b^3$ nodes

…

Level d has $b^{d-1}*b = b^d$ nodes
So the total number of nodes is:

$N = 1 + b + b^2 + b^3 + \ldots + b^d$ so therefore
$bN = b + b^2 + b^3 + \ldots + b^d + b^{d+1}$ so subtracting the 2nd line from the first, we have:

$(1-b)N = b^{d+1} - 1$ or $N = (b^{d+1} - 1)/b-1$ which is $O(b^d)$



Performance of BFS…

4. Time complexity –
   - In the worst case, the Goal will be at the far, right corner leaf of the search tree.

   - Processes all nodes above shallowest solution
   - Let depth of shallowest solution be d
   - Search takes time **O(b^d)**

Performance of BFS…

4. Time complexity –
   - In the worst case, the Goal will be at the far, right corner leaf of the search tree.

   - If the algorithm were to apply the goal test to nodes when selected for expansion, rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected. (i.e. the algorithm will have to search all the nodes in the tree, plus one extra level of b nodes to account for our stopping condition.). i.e.

$$b^d * b = O(b^{d+1}).$$

Uniform Cost Search (UCS)

- Breadth-first is only optimal if step costs is increasing with depth (e.g. constant). Can we guarantee optimality for any step cost?

- When all step costs are equal, **breadth-first** search is optimal because it always expands the shallowest unexpanded node.

- By a simple extension, we can find an algorithm that is optimal with *any step-cost* function, it is **Uniform Cost Search (UCS)**.

Uniform Cost Search (UCS)…

- Instead of expanding the shallowest node, **UCS** expands the node ***n*** with the lowest path cost ***g(n)***.
- **Uniform Cost Search** is the best algorithm for a search problem which does not involve the use of **heuristics**.
- It can solve any general graph for **optimal cost**.
- UCS uses a priority queue.
- UCS gives the minimum cumulative cost the maximum priority.

Uniform Cost Search…

Algorithm:

Insert the root into the priority-queue

While the queue is not empty

De-queue the maximum priority element from the queue

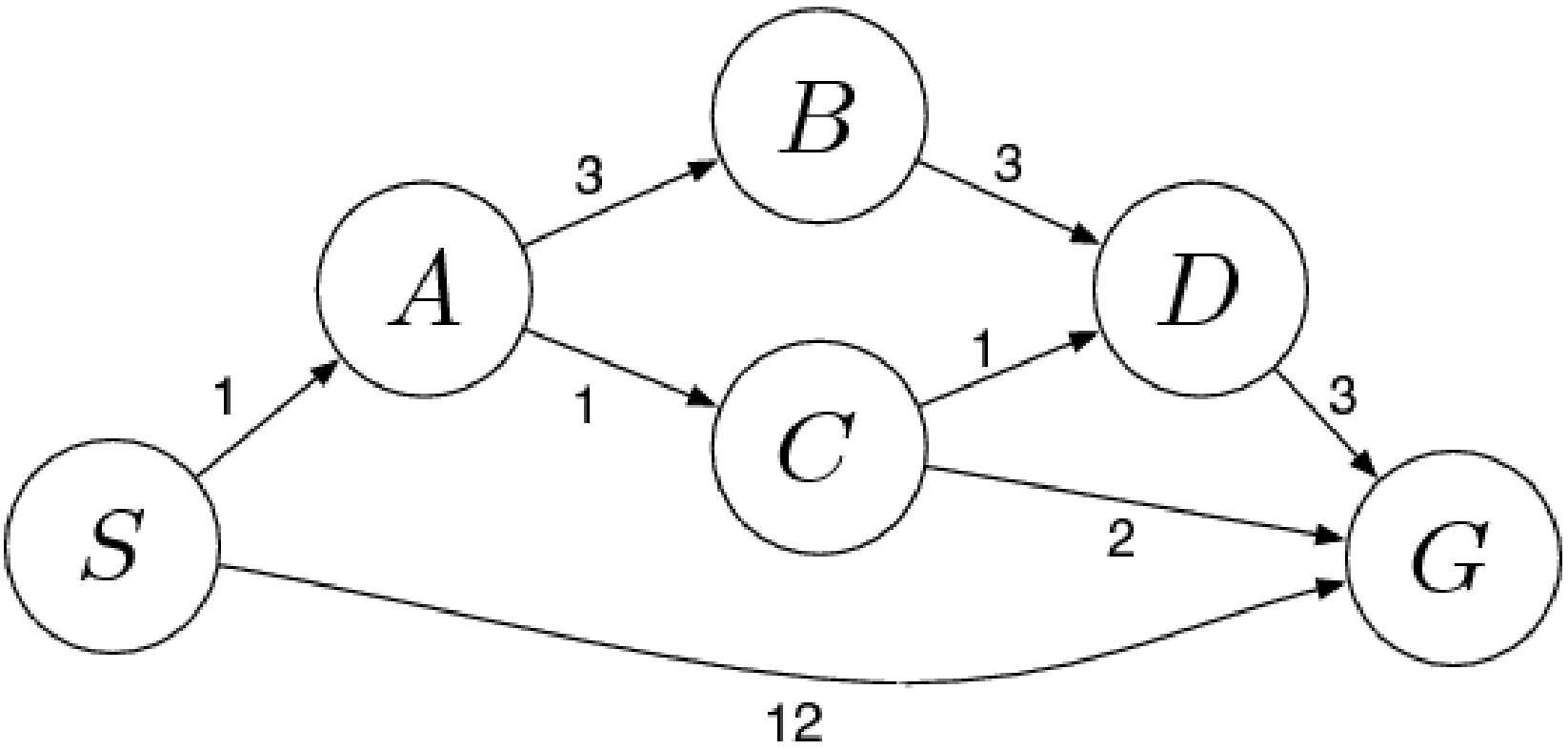
(If priorities are same, alphabetically smaller path is chosen)

If the path is ending in the goal state, print the path and exit

Else

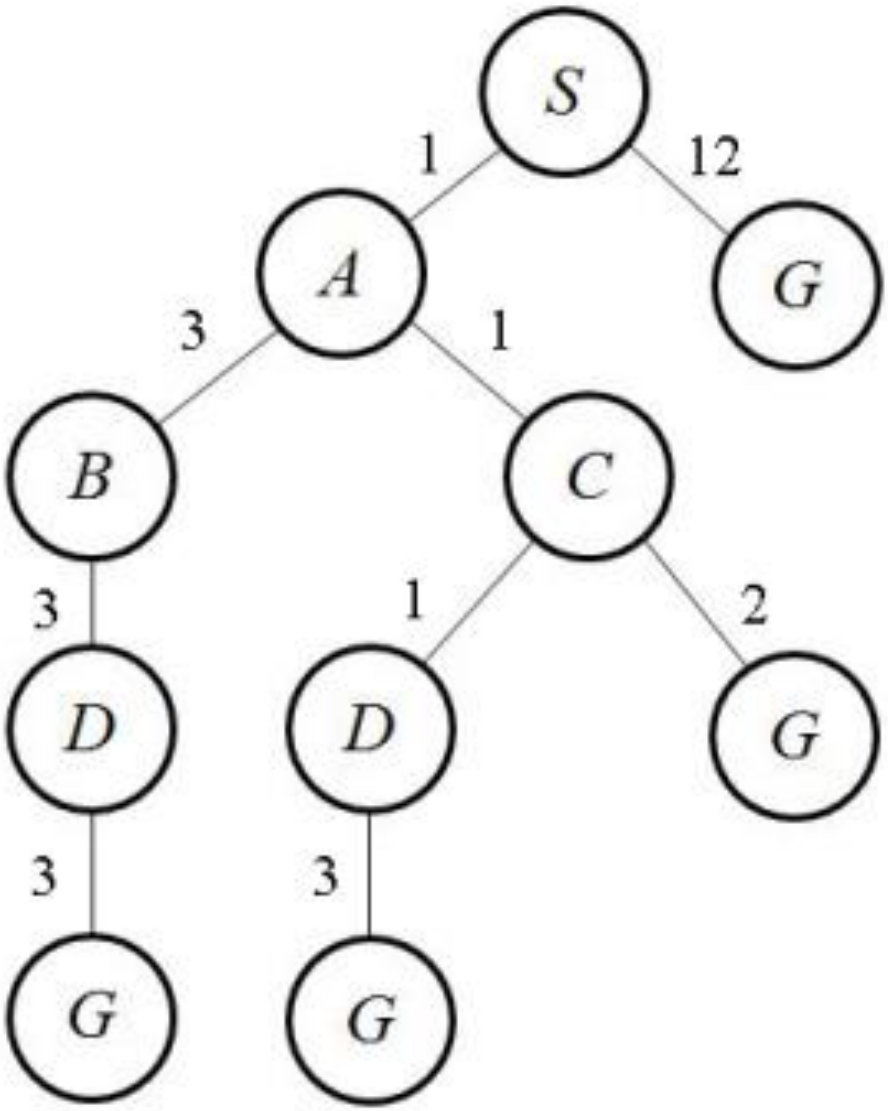
Insert all the children of the de-queued element, with the **cumulative costs as priority**.

Uniform Cost Search…



Uniform Cost Search…

Search Tree



https://algorithmicthoughts.wordpress.com/2012/11/13/artificial-intelligence-depth-first-search/

Uniform Cost Search…

Initialization: { [S , 0] }

Iteration1: { [S->A , 1] , [S->G , 12] }

Iteration2: { [S->A->C , 2] , [S->A->B , 4] , [S->G , 12] }

Iteration3: { [S->A->C->D , 3] , [S->A->B , 4] , [S->A->C->G , 4] , [S->G , 12] }

Iteration4: { [S->A->B , 4] , [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->G , 12] }

Iteration5: { [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->A->B->D , 7] , [S->G , 12] }

Iteration6: Gives the final output as **S->A->C->G.**

Uniform Cost Search…

- At any given point in the execution, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the graph.

Performance of UCS

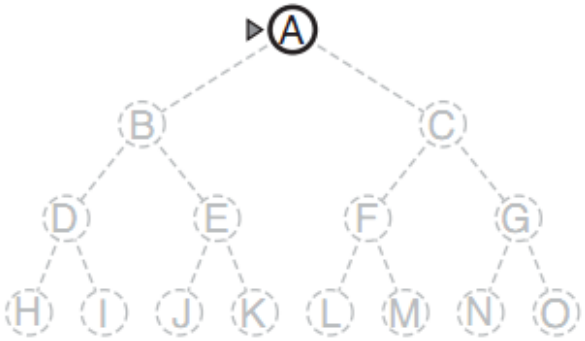
- Optimality: Based on *Graph Separation Theory* and *Non-negative Step-cost*, UCS is optimal.

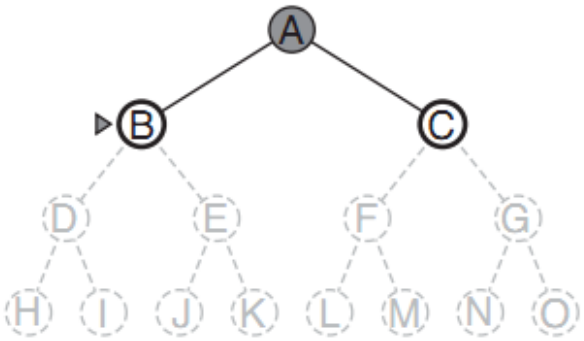
- Completeness: Uniform-cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost. Completeness is guaranteed provided the cost of every step exceeds some small positive constant ϵ.

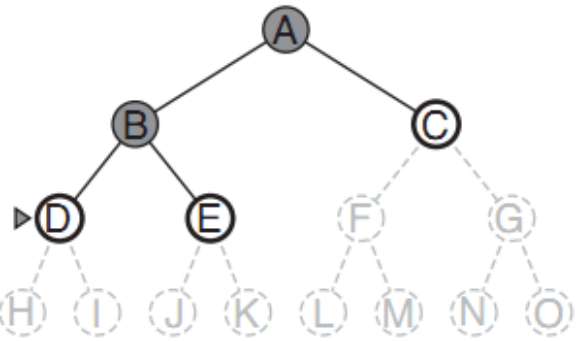
Performance of UCS…

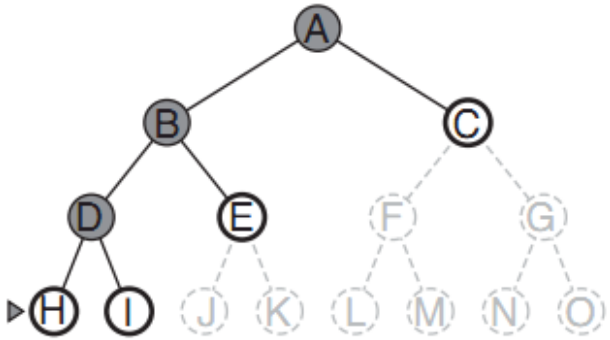
- Complexity: UCS is guided by path costs rather than depths, so its complexity is not easily characterized in terms of ***b*** and ***d***. Instead, let C^* be the cost of the optimal solution, and assume that every action costs at least ϵ. Then the algorithm's ***worst-case time*** and ***space complexity*** is $O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$. When all step costs are equal, $O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$ is just b^{d+1}.

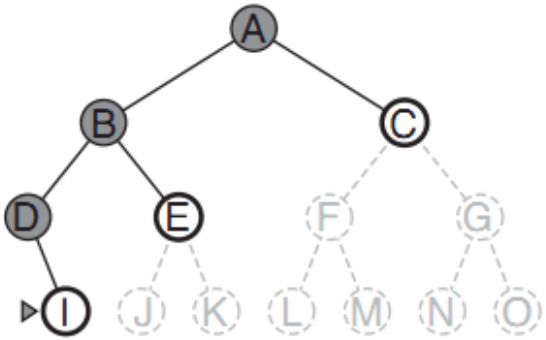
Depth First Search (DFS)

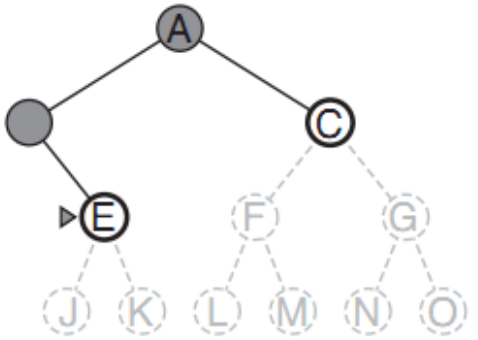




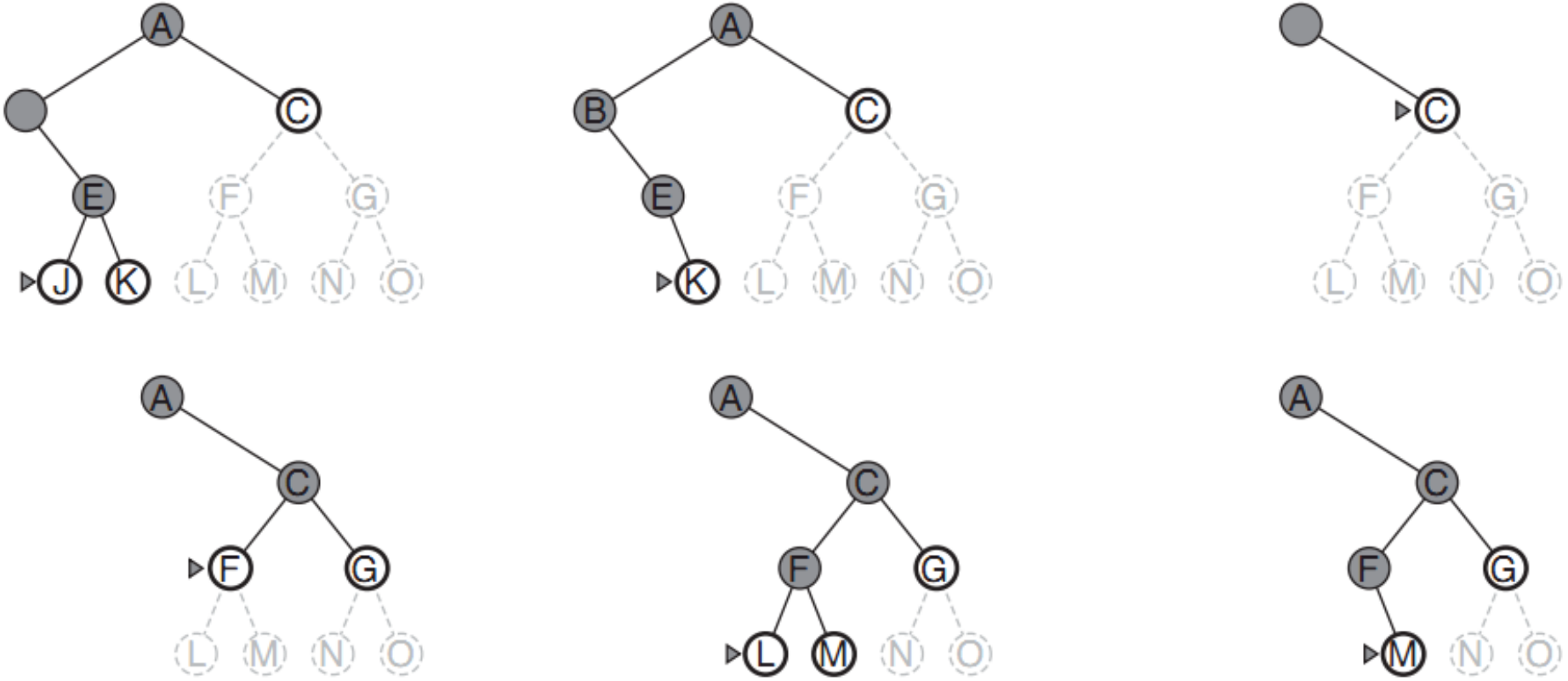








Depth First Search (DFS)…



Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Performance of DFS

- Optimality: Not optimal.
- Completeness: In Tree version search, DFS is not complete.
- Time Complexity: **O(b^d)**

Performance of DFS…

- Space Complexity: A depth-first tree search needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored. For a state space with branching factor b and depth d, depth-first search requires storage of only $O(bd)$ nodes.

Limitation of DFS

- Non-termination: When the length of a path in the search tree is infinite, non-termination occurs. So, we perform DFS to a limited depth which is called **Depth Limited Search**.

Depth-Limited Search (DLS)

- Depth is limited to a certain value; E.g. $l = 3$.
- Depth limit solves the infinite-path problem.
- Depth limits can be based on knowledge of the problem.

- Depth-first search can be viewed as a special case of depth-limited search with $l = \infty$

Performance of DLS

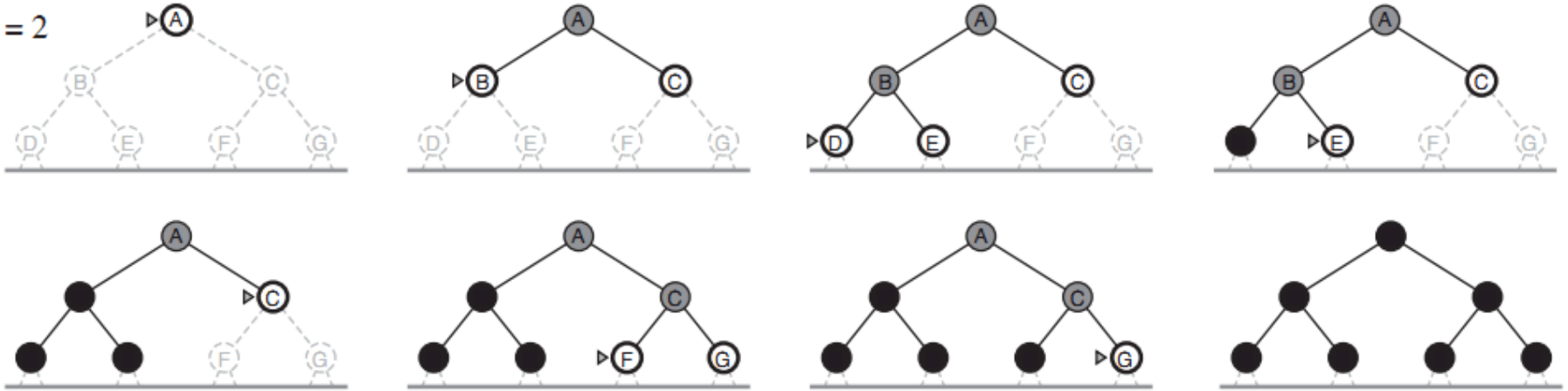
- Optimality: Not optimal, though $l > d$, because DFS is not optimal itself.
- Completeness: If $l < d$, DLS may not find the goal. Hence, incomplete.
- Time Complexity: **O(b^*l*)**
- Space Complexity: **O(b*l*)**

Iterative Deepening Search

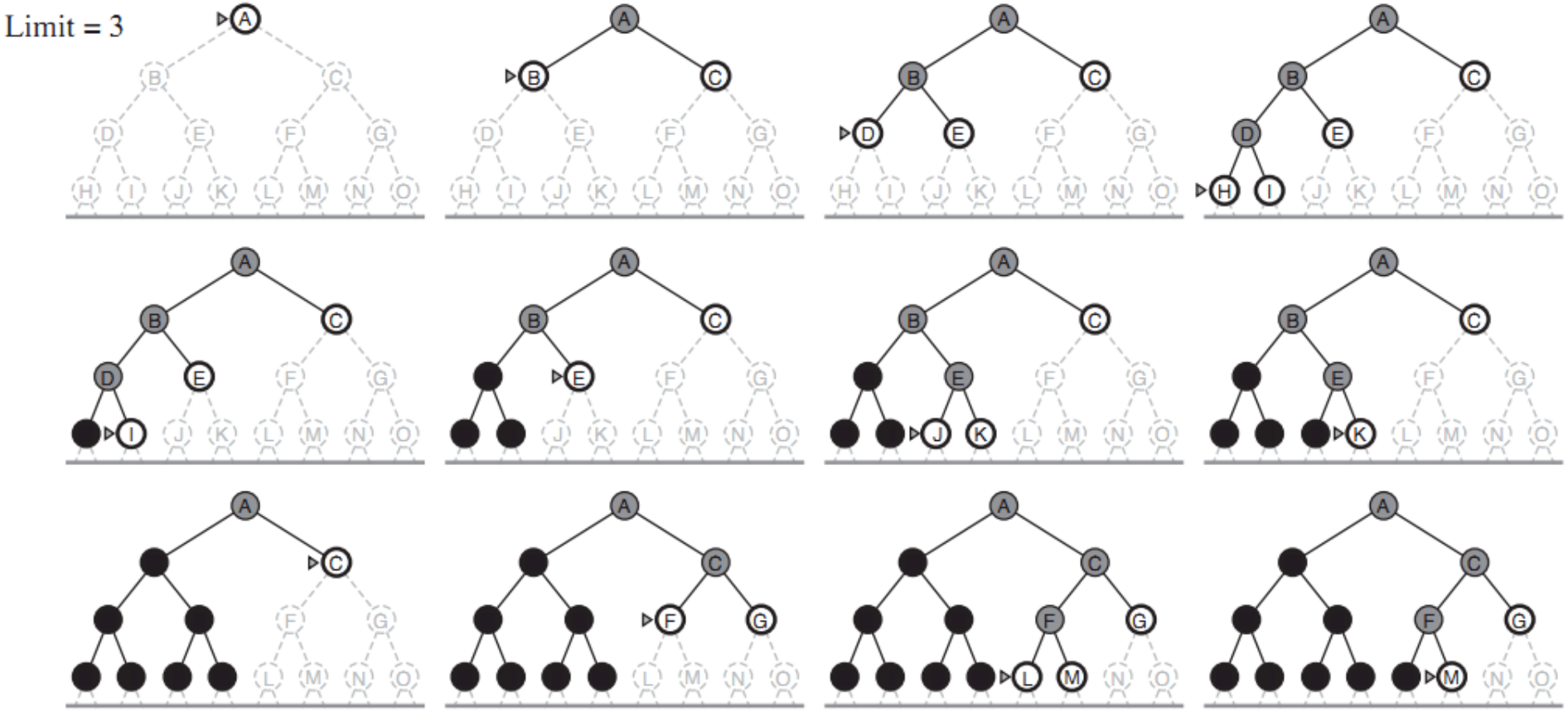
Limit = 0

Limit = 1

Limit = 2



Iterative Deepening Search…



Informed (Heuristic) Search

- **General Sense** – Any advice that is often effective, but is not guaranteed to work in every case.

- Heuristic search is an AI search technique that employs heuristic i.e. **rule of thumbs** that probably leads to a solution.

- Most of the search problems have exponential nature. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number.

- A technique is designed to solve a problem more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution.

Objectives & Motivation

- To produce a solution in a **reasonable time frame** that is good enough for solving the problem.

- This **solution may not be the best** but it is valuable because finding it does not require a prohibitively long time.

Heuristic Function/ Heuristic Evaluation Function/ Heuristic

- Heuristics basically predict how far the goal state may be or how much it will cost to get to the goal state from a particular node.
- Is a way to inform the search about the direction to a goal.
- It provides an informed way to guess which neighbor of a node will lead to a goal.
- Estimates the cost of an optimal path between a pair of states in a single-agent path finding problem.
- **Key-Properties:** It estimates actual cost and inexpensive to compute.

Heuristic Information

- Information about the problem include the **nature of states**, cost of transforming from one state to another and **characteristics of the goals**.
- Heuristic information can be expressed in the form of Heuristic Evaluation Function, say ***f(n, g)***; a function of the nodes ***n*** and/or the goals ***g***.
- Searching techniques that uses such heuristic function are as:
  - ✓ **A* Algorithm**
  - ✓ **Hill-Climbing**
  - ✓ **Simulated Annealing**

Best First Search

- In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function.
- The idea of **Best First Search** is to **use an evaluation function** ***f(n)*** to decide which adjacent is most promising and then explore.
- **Best First Search** falls under the category of Heuristic Search or Informed Search.
- This "best first" behavior is implemented with a ***Priority-Queue.***

Best First Search…

- Best-first search tries to minimize that number of expanded nodes utilizing Path Cost with Heuristic Function.

- An example of best-first search is e.g. ***A* algorithm.***

A* Search Algorithm

- A* is a **path finding** algorithm.
- A* is an informed search or best-first search i.e. it solves problems by searching among all most promising paths to the solution (goal).
- A* uses a modified evaluation function.

A* Search Algorithm…

- A* minimizes the total path cost. Under the right condition, A* provides the cheapest cost solution in the optimal path.

- The evaluation function '***f***' is an estimate of the value of a node ***x***, given by:

$$f(n) = g(n) + h(n)$$

Here, g(n) → The cost to get from the **start state** to **state n**.

h(n) → The estimated cost to get from **state n** to the **goal state**. This is often referred to as the heuristic, which is nothing but a kind of smart guess (the heuristic).

- Loops are avoided. The same state is not expanded twice.

A* Search Algorithm…

Algorithm:

Insert the root node into the priority-queue

While the queue is not empty

De-queue the element with the highest priority

(If priorities are same, alphabetically smaller path is chosen)

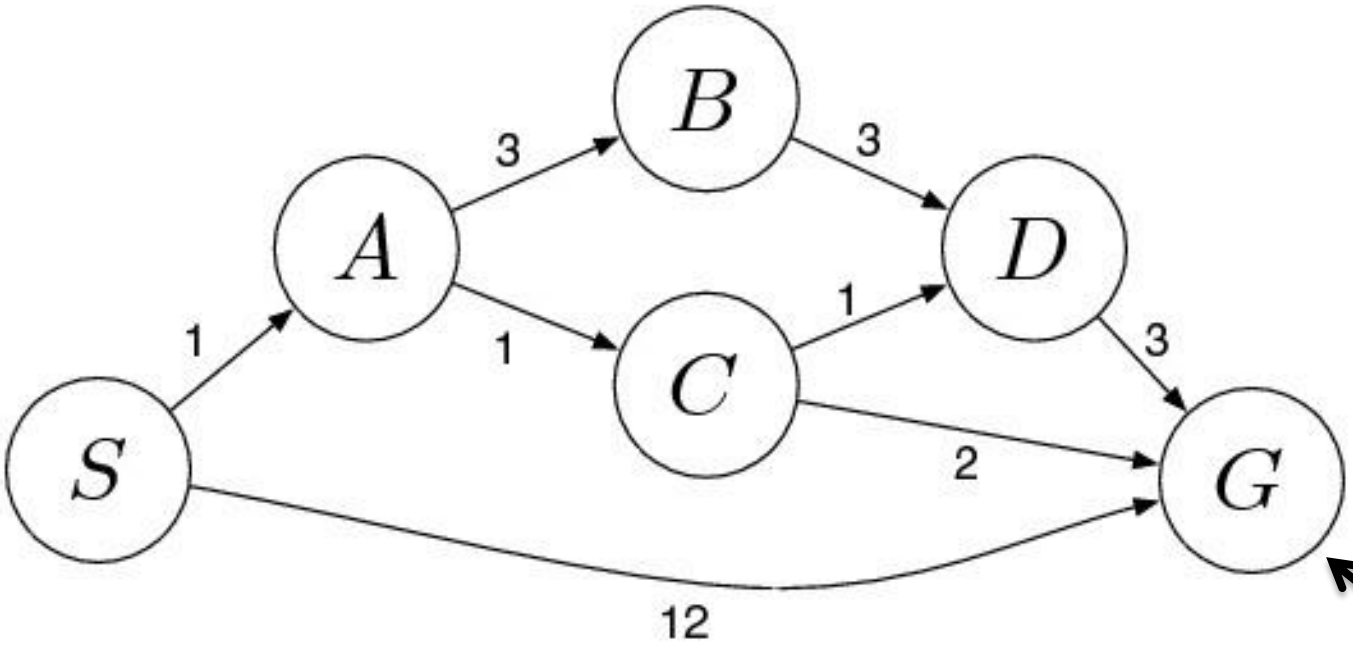
If the path is ending in the goal state, print the path and exit

Else

Insert all the children of the de-queued element, with **f(n) as the priority**

Note: The algorithm returns a path which is optimal in terms of cost, if an **admissible heuristic** is used.

A* Search Algorithm…



State	h1	h2
S	4	3
A	2	2
B	6	5
C	2	1
D	3	2
G	0	0

Goal

A* Search Algorithm…

Initialization: { [S , 4] }

---

f(S) = g(S) + h(S) = 0 + 4 = 4

Iteration1: { [S->A , 3] , [S->G , 12] }

---

f(A) = g(A) + h(A) = 1+2=3; f(A) < f(G)

f(G) = g(G) + h(G) = 12+0=12; Hence, priority(A)>priority(G)

Iteration2: { [S->A->C , 4] , [S->A->B , 10] , [S->G , 12] }

---

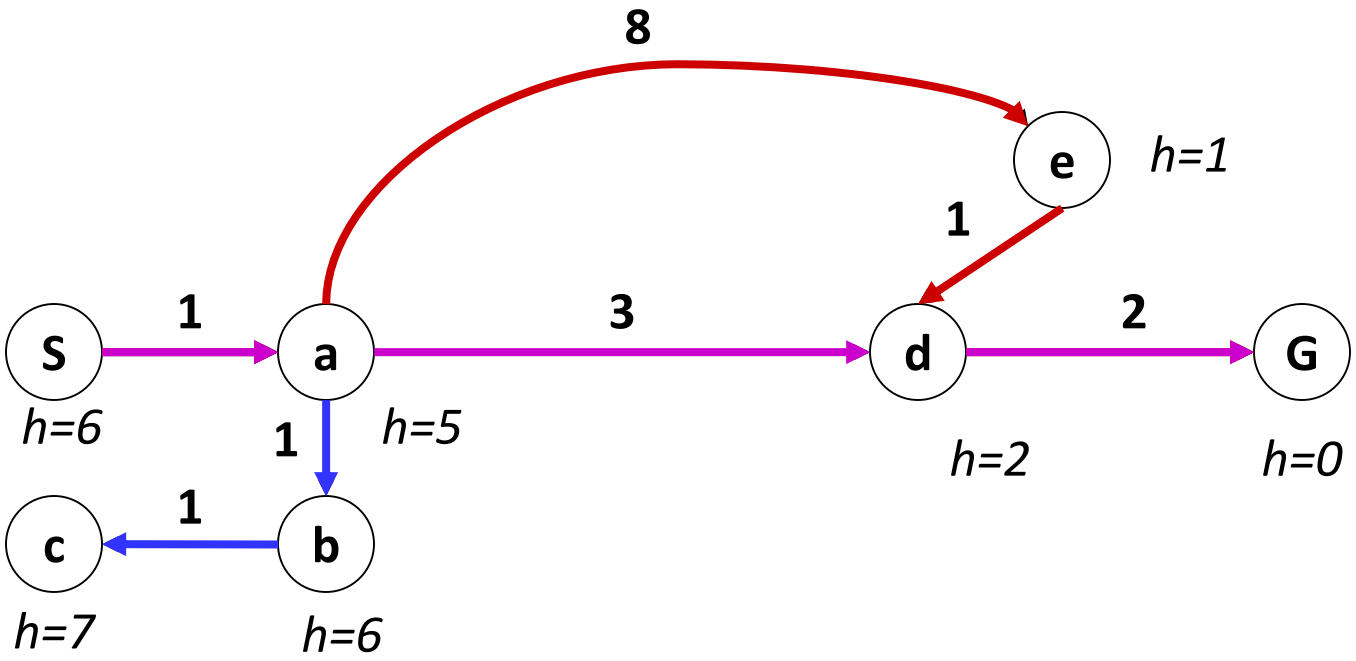
f(B) = g(B) + h(B) = 1+3+6 = 10; f(C) < f(B)

f(C) = g(C) + h(C) = 1+1+2=4; Hence, priority(C)>priority(B)

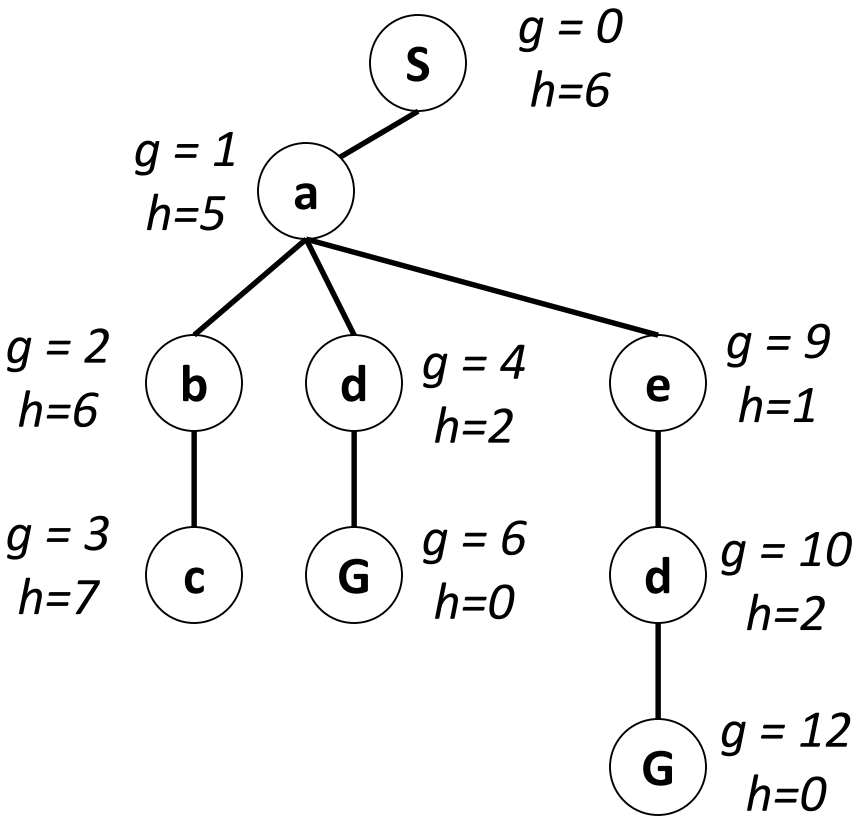
Iteration3: { [S->A->C->G , 4] , [S->A->C->D , 6] , [S->A->B , 10] , [S->G , 12] }

Iteration4: Gives the final output as S->A->C->G.

A* Search Algorithm…



A* Search Algorithm…



A* Search Algorithm…

Limitations:

Although being the best path finding algorithm, A* Search Algorithm doesn't produce the shortest path always, as it relies heavily on heuristics/approximations to calculate – h

Performance of A* Search

Optimality:

- The tree-search version of A∗ is optimal if h(n) is admissible.
- Admissible heuristic is one that never overestimates the cost to reach the goal.
- Because g(n) is the actual cost to reach n along the current path, and f (n) = g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n.

Performance of A* Search…

Completeness:

- The number of nodes is finite.
- Completeness is guaranteed provided the cost of every step exceeds some small positive constant ϵ.

Hill Climbing Search

- One of the most simplest procedures for implementing heuristic search.

- Hill climbing search algorithm is simply **a loop** that continuously moves in the direction of increasing value, i.e. uphill. It stops when it reaches a '**peak**' where no neighbor has higher value.

- Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space.

Hill Climbing Search…

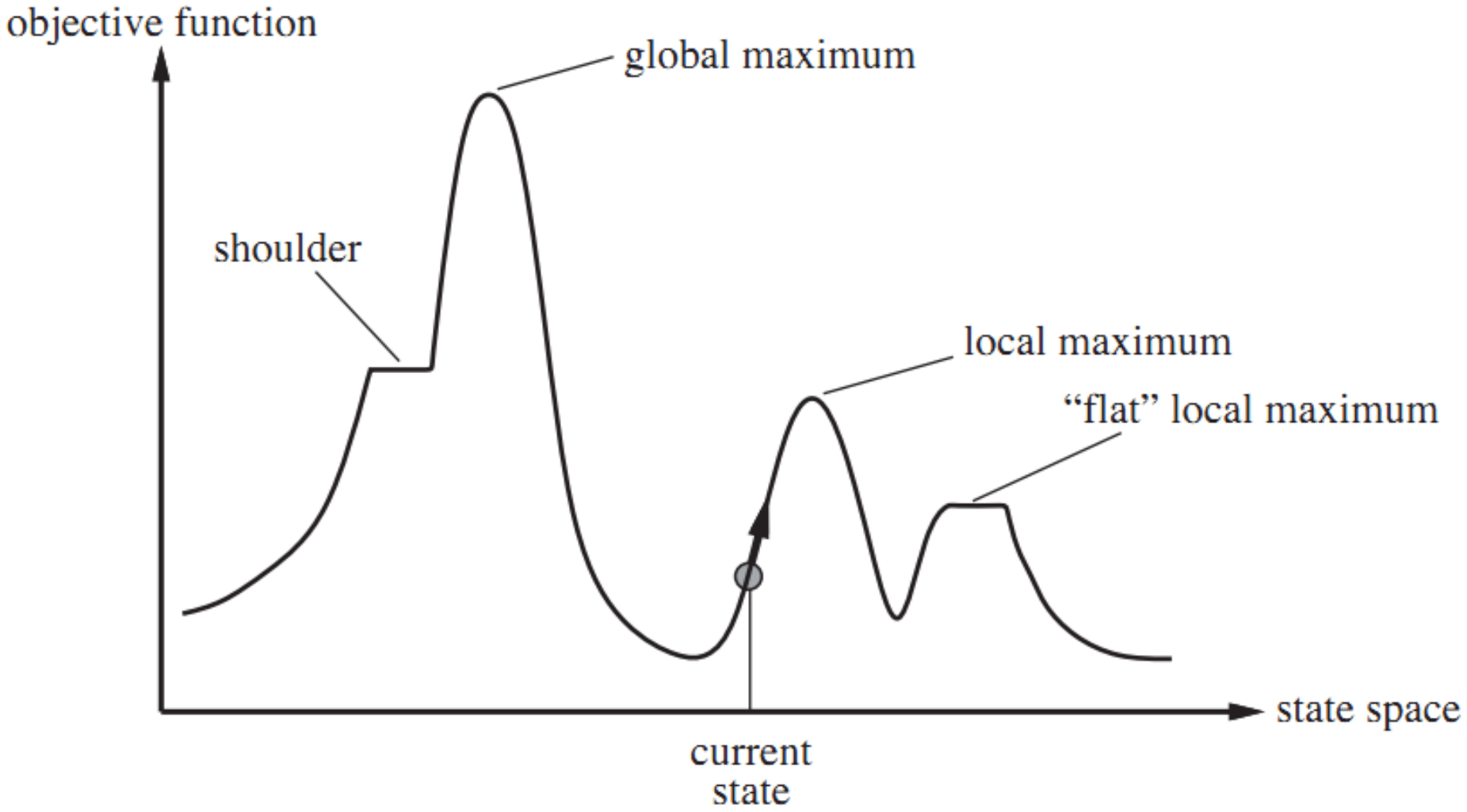


Fig. 01: State-space Landscape

Hill Climbing Search…

- **Local maximum:** A local maximum is a solution that surpasses other neighboring solutions or states but is not the best possible solution.
- **Global maximum:** This is the best possible solution achieved by the algorithm.
- **Current state:** This is the existing or present state.
- **Flat local maximum:** This is a flat region where the neighboring solutions attain the same value.
- **Shoulder:** This is a plateau whose edge is stretching upwards.

Hill Climbing Search…

- A ***State-space Landscape*** can be used to describe Hill Climbing Search.

- A ***Landscape*** has
  - ✓ Location (defined by the state such as Current State)
  - ✓ Elevation (defined by the value of the heuristic cost function or objective function)

- If elevation corresponds to cost, then the aim is to find the *lowest valley – a global minimum*;

- If elevation corresponds to an objective function, then the aim is to find the *highest peak – a global maximum*.

Hill Climbing Algorithm

Step-1: Evaluate the starting state. If it is a goal state then stop and return success.

Step-2: Else, continue with the starting state as considering it as a current state.

Step-3: Continue Step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.

Step-4:

a. Select a state that has not been yet applied to the current state and apply it to produce a new state.

b. Function to evaluate a new state.

   i. If the current state is a goal state then stop and return success.

   ii. If new state is better than the current state then make it current state and proceed further.

   iii. If not better then continue in the loop until a solution is found.

Step-5: Exit

A Nutrition Problem

- A nutritionist advises an individual who is suffering from iron and vitamin B deficiency to take at least 2400 milligrams (mg) of iron, 2100 mg of vitamin B_1, and 1500 mg of vitamin B_2 over a period of time.
- Two vitamin pills are suitable, brand-A and brand-B.
- Each brand-A pill costs 6 cents and contains 40 mg of iron, 10 mg of vitamin B_1, and 5 mg of vitamin B_2.
- Each brand-B pill costs 8 cents and contains 10 mg of iron and 15 mg each of vitamins B_1 and B_2.
- What combination of pills should the individual purchase in order to meet the minimum iron and vitamin requirements at the lowest cost?

A Nutrition Problem

Solution

- **In short, we want to minimize the objective function**

$$C = 6x + 8y$$

subject to the system of inequalities

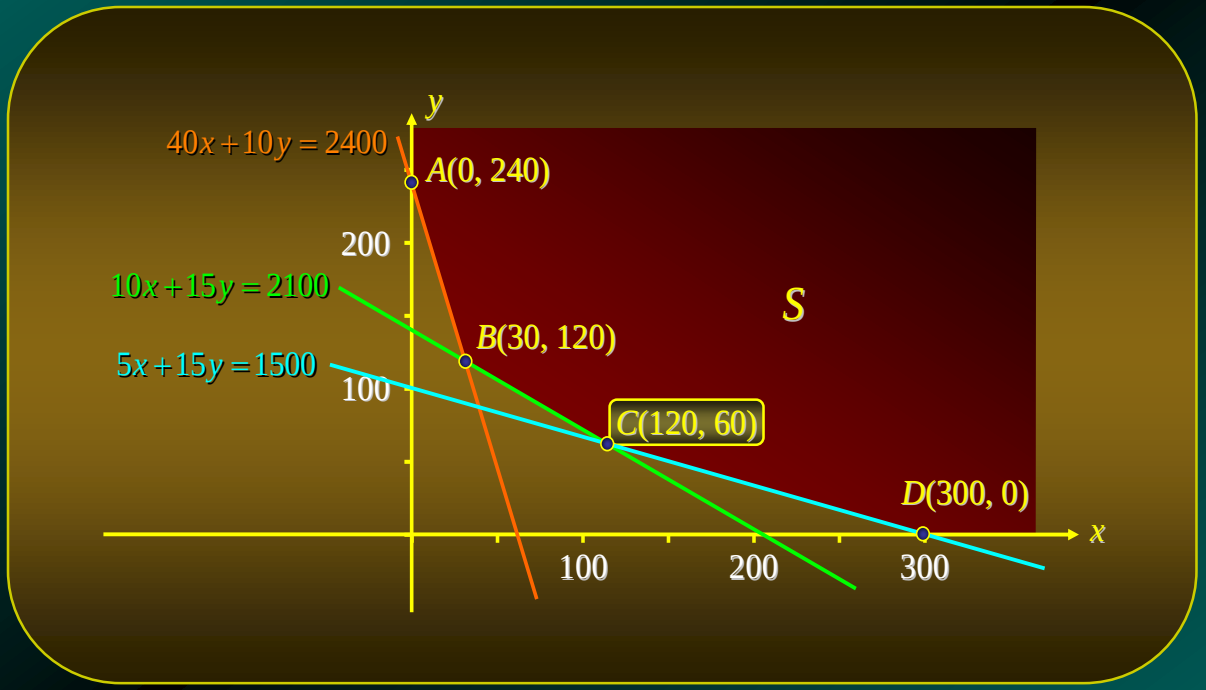
$$40x + 10y \geq 2400$$

$$10x + 15y \geq 2100$$

$$5x + 15y \geq 1500$$

$$x \geq 0$$

$$y \geq 0$$


y
40x + 10y = 2400
A(0, 240)
200
10x + 15y = 2100
S
B(30, 120)
5x + 15y = 1500
100
C(120, 60)
D(300, 0)
x
100
200
300

Advantages of Hill Climbing

- Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management.

- It is also helpful to solve pure optimization problems (e.g. Nutrition Problem) where the objective is to find the best state according to the objective function.

- It requires much less conditions than other search techniques.

Disadvantages of Hill Climbing

Local Maxima:

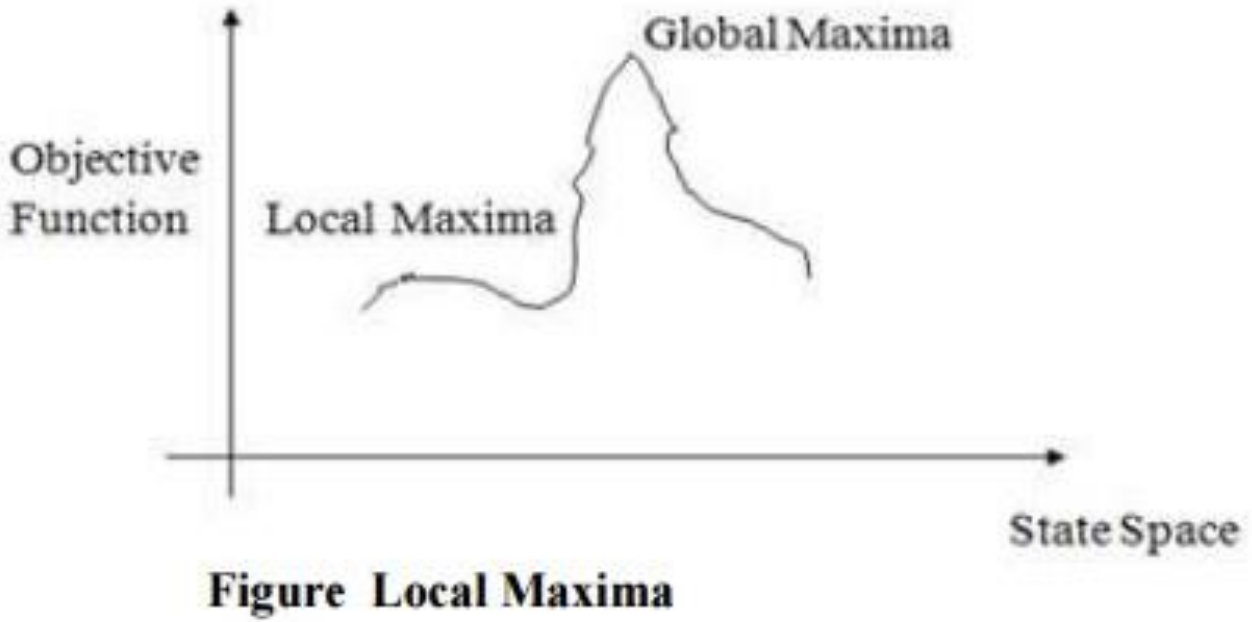


Figure Local Maxima

Disadvantages of Hill Climbing

Local Maxima: At this point, the neighboring states have lower values than the current state. The greedy approach feature will not move the algorithm to a worse off state. This will lead to the hill-climbing process's termination, even though this is not the best possible solution.

Solution: This problem can be solved using momentum. This technique adds a certain proportion (m) of the initial weight to the current one. m is a value between 0 and 1. Momentum enables the hill-climbing algorithm to take huge steps that will make it move past the local maximum.

Disadvantages of Hill Climbing…

Ridges:

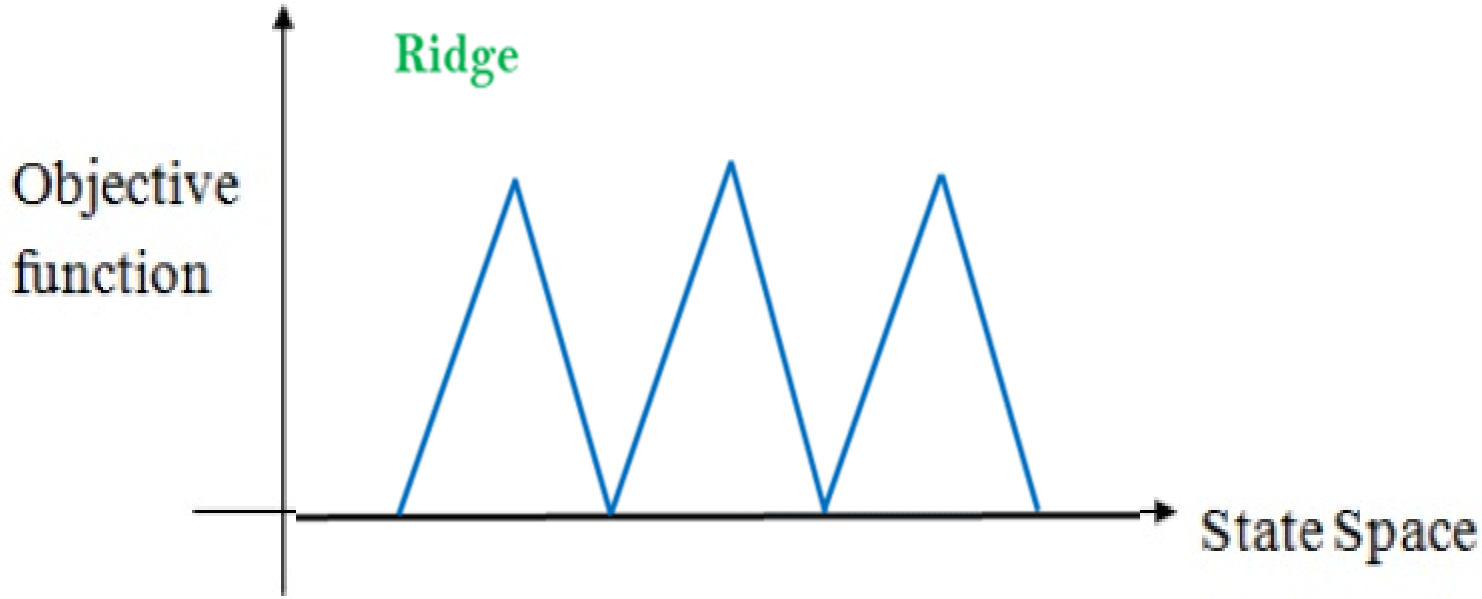


Figure Ridges

Disadvantages of Hill Climbing…

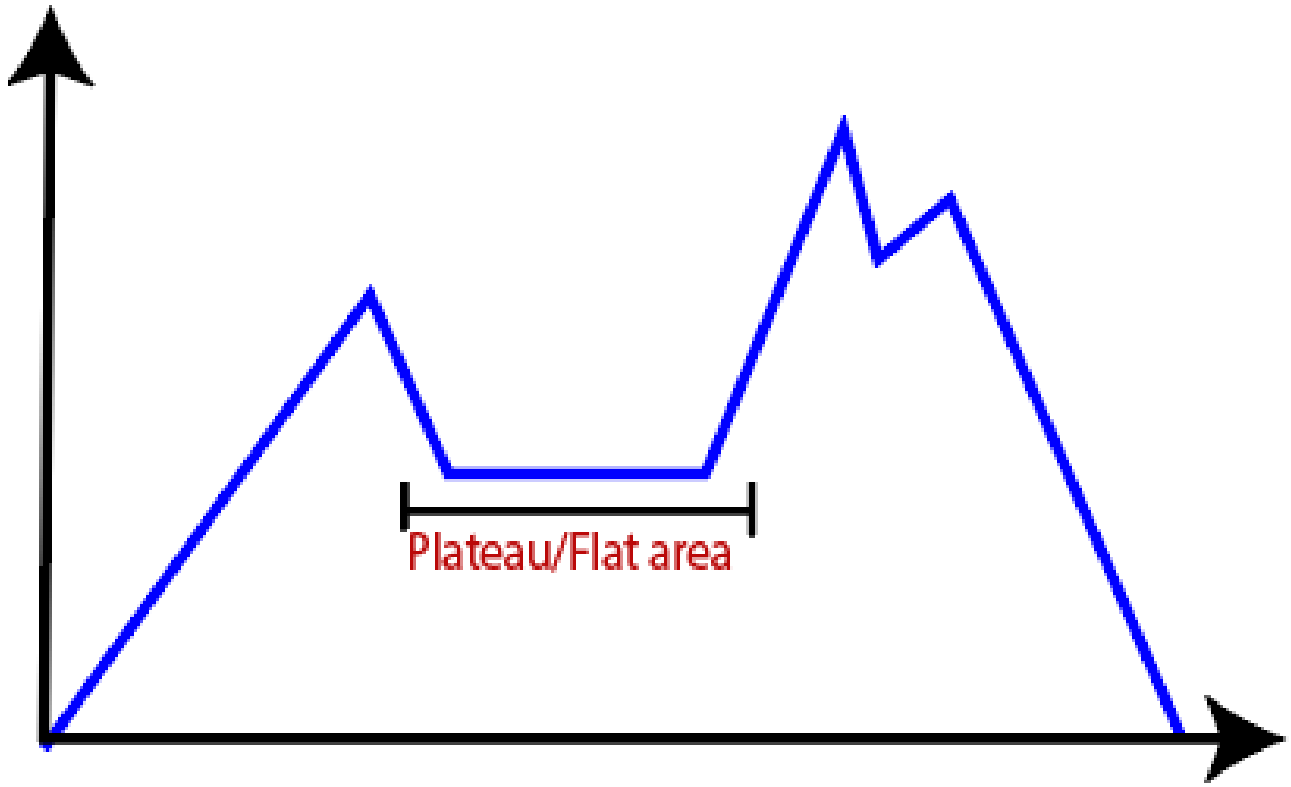
Ridges: The hill-climbing algorithm may terminate itself when it reaches a ridge. This is because the peak of the ridge is followed by downward movement rather than upward movement.

Solution: This impediment can be solved by going in different directions at once.

 https://www.section.io/engineering-education/understanding-hill-climbing-in-ai/

Disadvantages of Hill Climbing…

Plateau:



Disadvantages of Hill Climbing…

Plateau: In this region, the values attained by the neighboring states are the same. This makes it difficult for the algorithm to choose the best direction.

Solution: This challenge can be overcome by taking a huge jump that will lead you to a non-plateau space.

Constraints Satisfaction Problem (CSP)

Adversarial Search

- Examining the problems that arise when we try to plan ahead in a world where other agents are planning against us.

Games Versus Search

Search – no adversary

- –Solution is (heuristic) method for finding goal
- –Heuristics and CSP techniques can find *optimal* solution

Evaluation function: estimate of cost from start to goal through given node.

Examples: path planning, scheduling activities

Games – adversary

- –Solution is strategy (strategy specifies move for every possible opponent reply).
- –Time limits force an *approximate* solution to be taken.

Evaluation function: evaluate "goodness" of game position

Examples: chess, checkers, Othello, backgammon

Games Search Problem

- Problem formulation
  - Initial state: initial board position + whose move it is
  - Operators: legal moves a player can make
  - Goal (terminal test): game over?
  - Utility (payoff) function: measures the outcome of the game and its desirability

- Search objective:
  - Find the sequence of player's decisions (moves) **maximizing its utility (payoff)**
  - **Consider the opponent's moves and their utility**

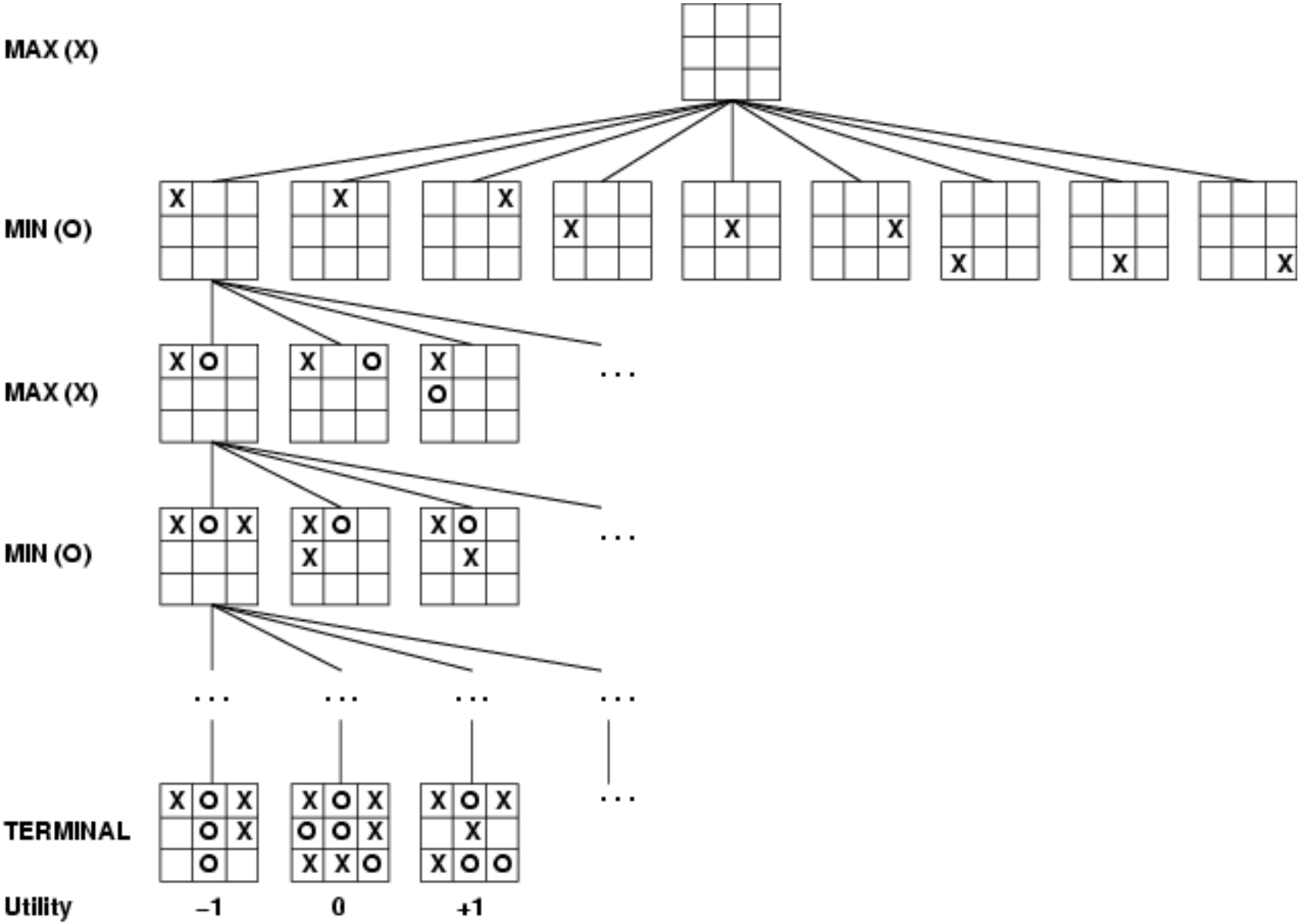
Example of An Adversarial 2 Person Game: Tic-Tac-Toe

- Player (x) moves first, then the Opponent (O).
- A two player game where MinMax algorithm is applied.
- In this game in order to win we must fill a row, a column or diagonal.

Objectives:

- Player: maximize outcome
- Opponent: minimize outcome

Tic-Tac-Toe…



MiniMax Algorithm

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

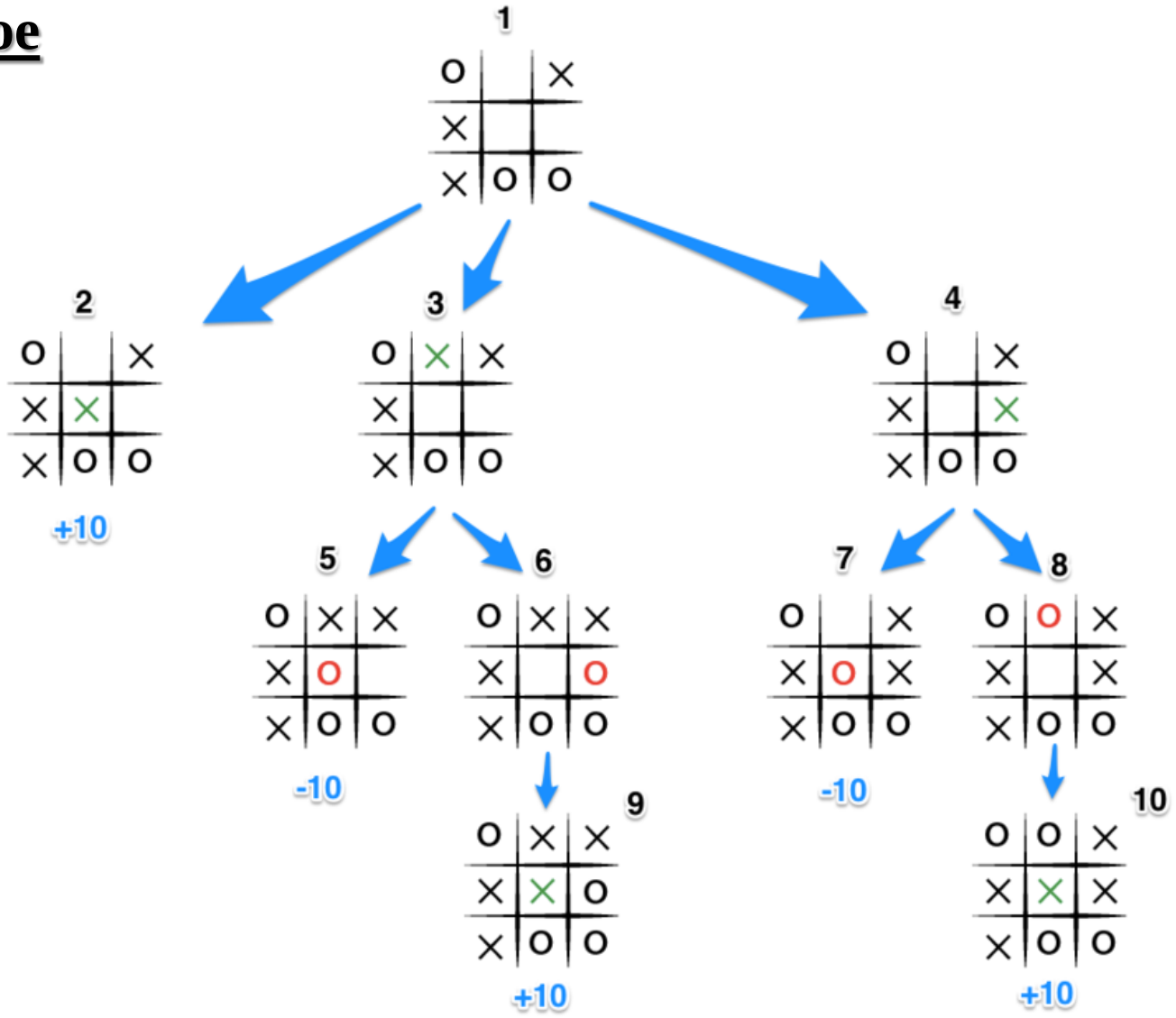
Figure: An algorithm for calculating minimax decisions.

MiniMax Algorithm…

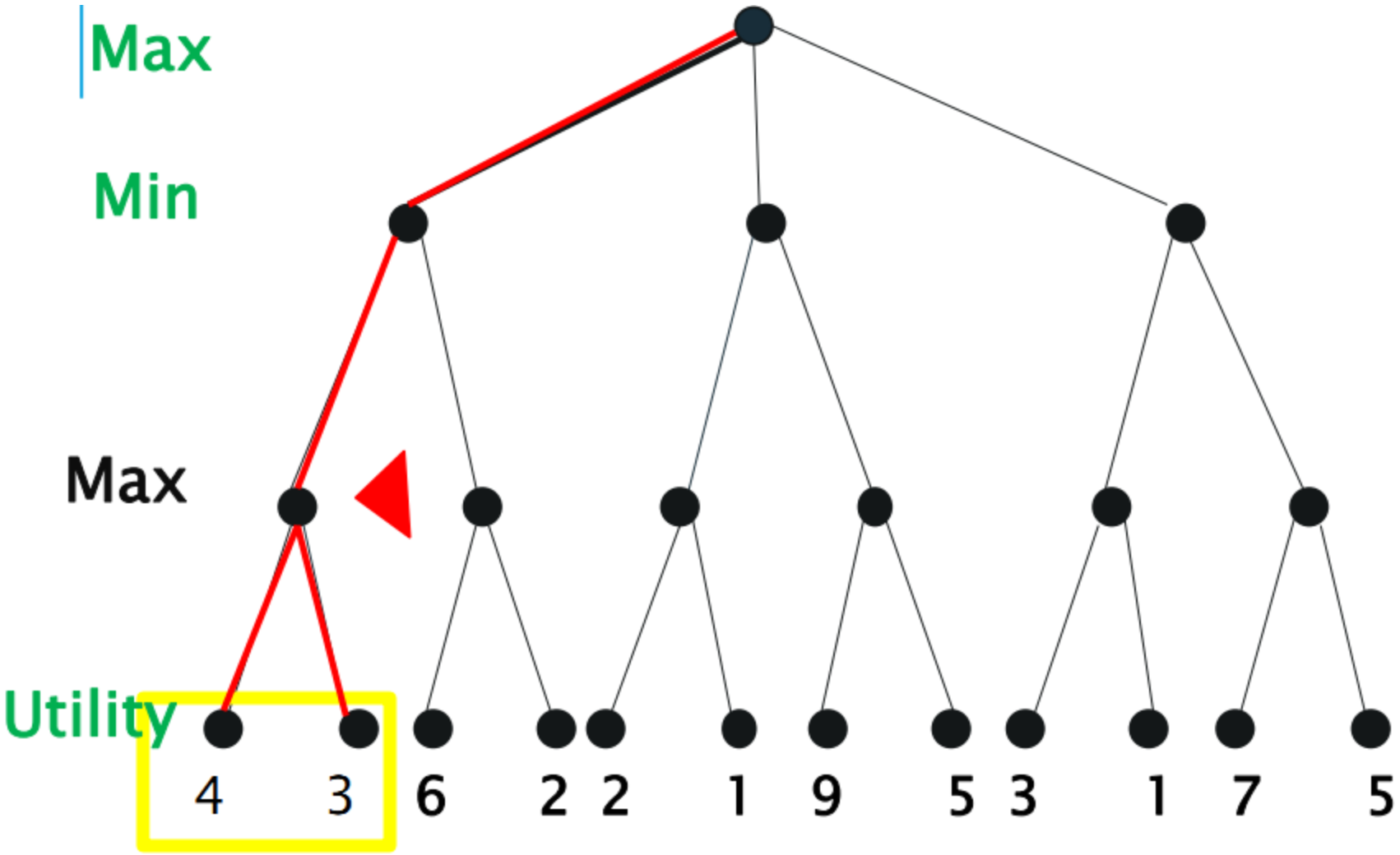
- First move Max (Player), then the Min (Opponent)
- By this way generate the game tree.

MiniMax Algorithm…

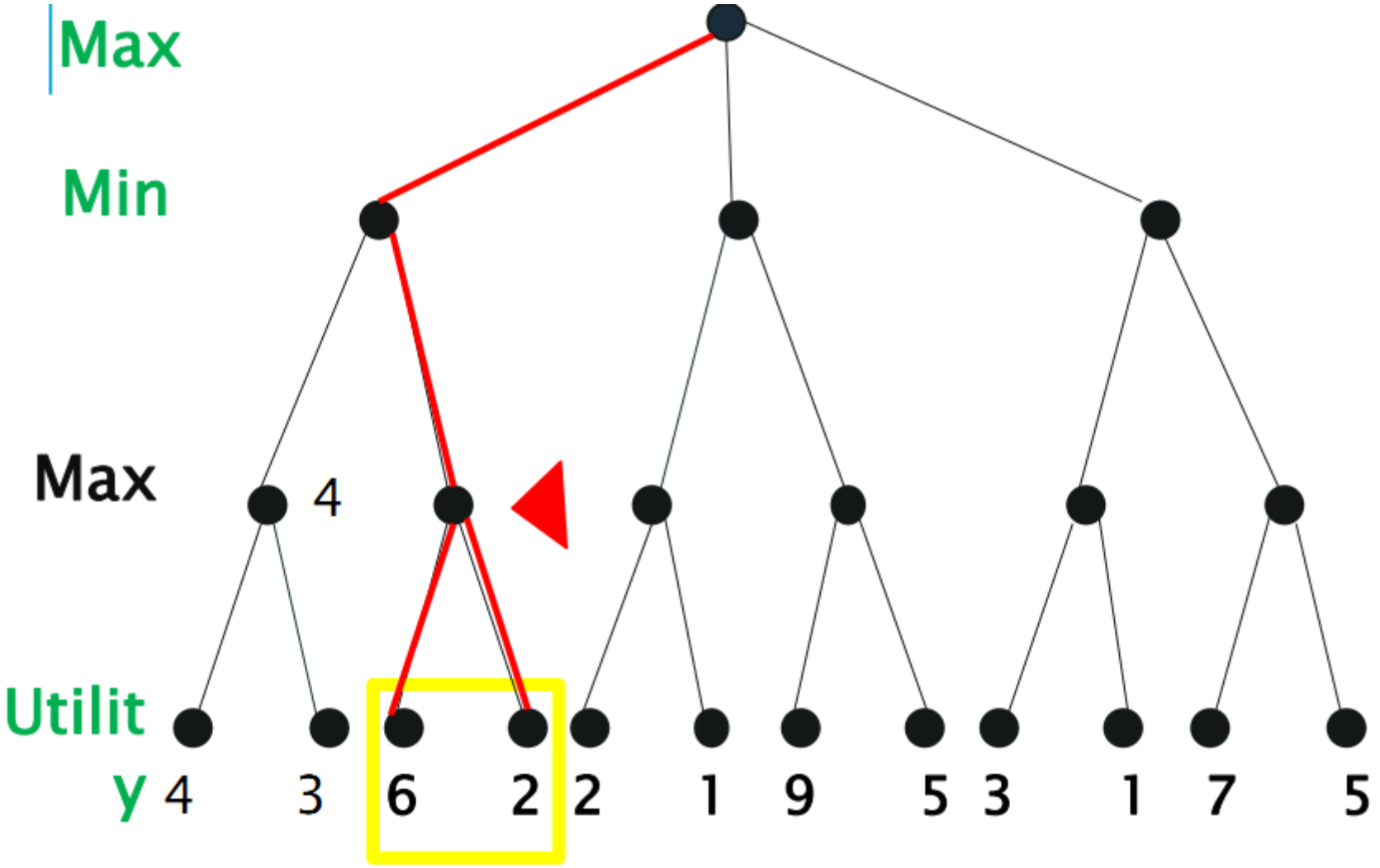
Tic-Tac-Toe



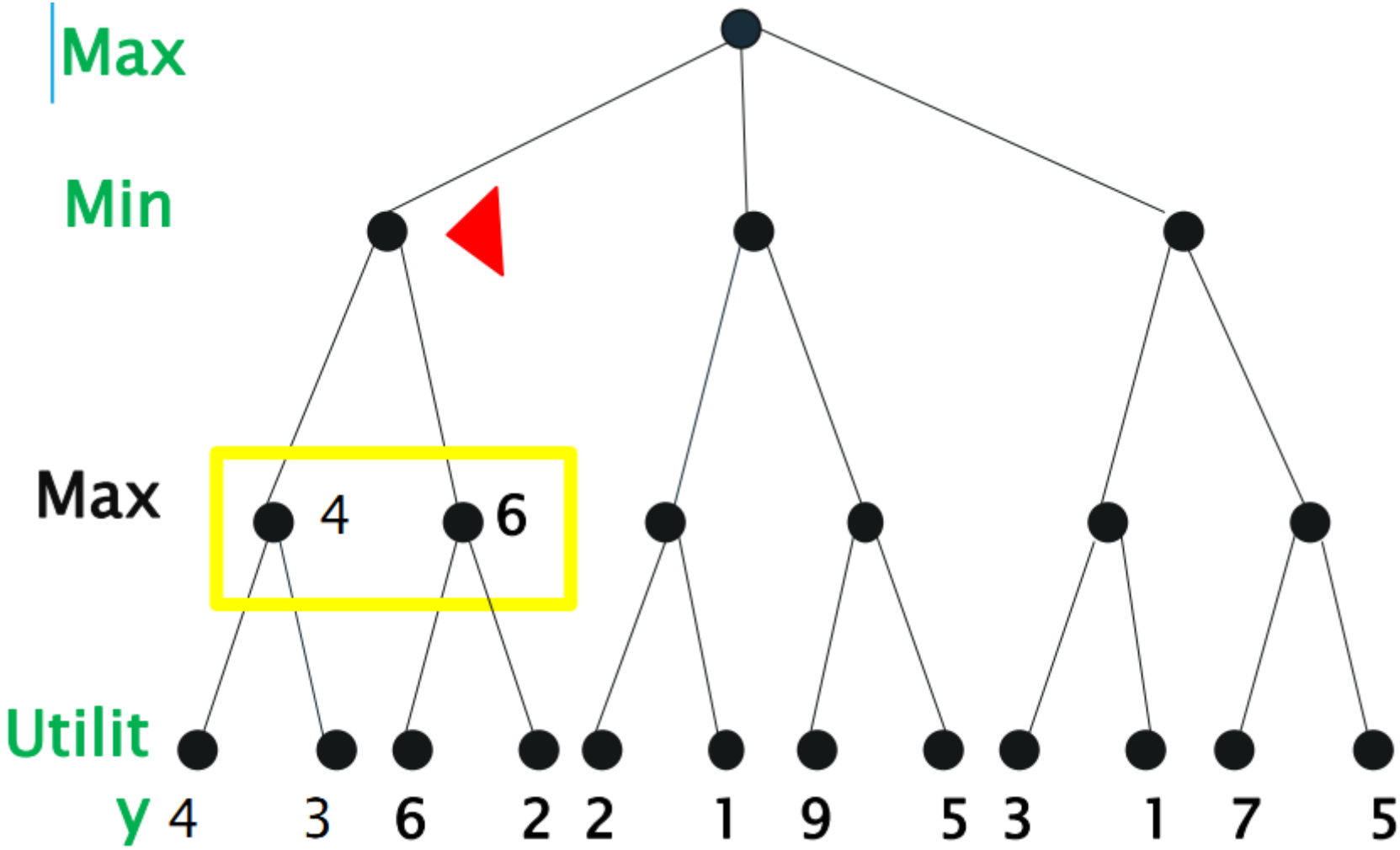
MiniMax Algorithm…



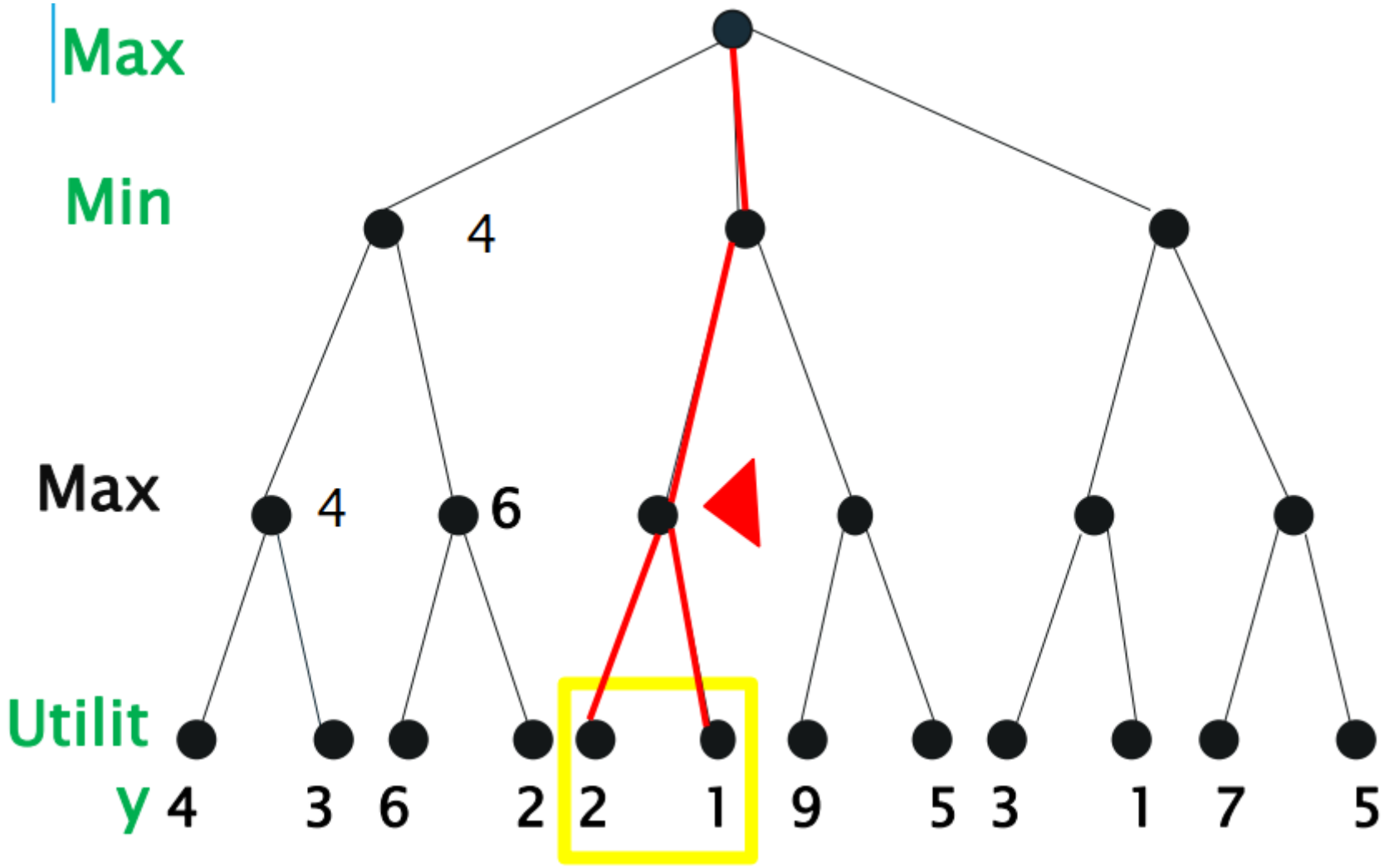
MiniMax Algorithm…



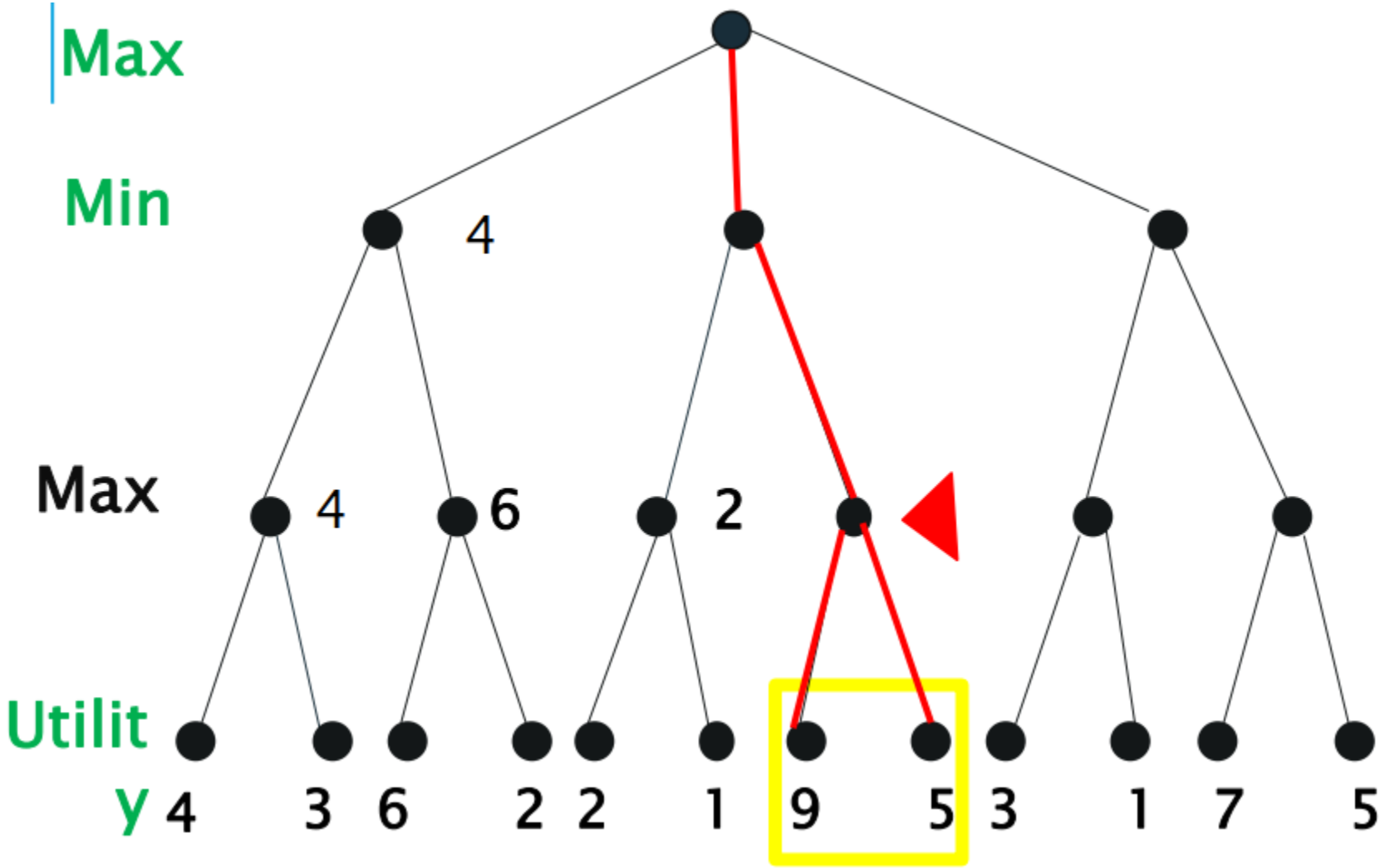
MiniMax Algorithm…



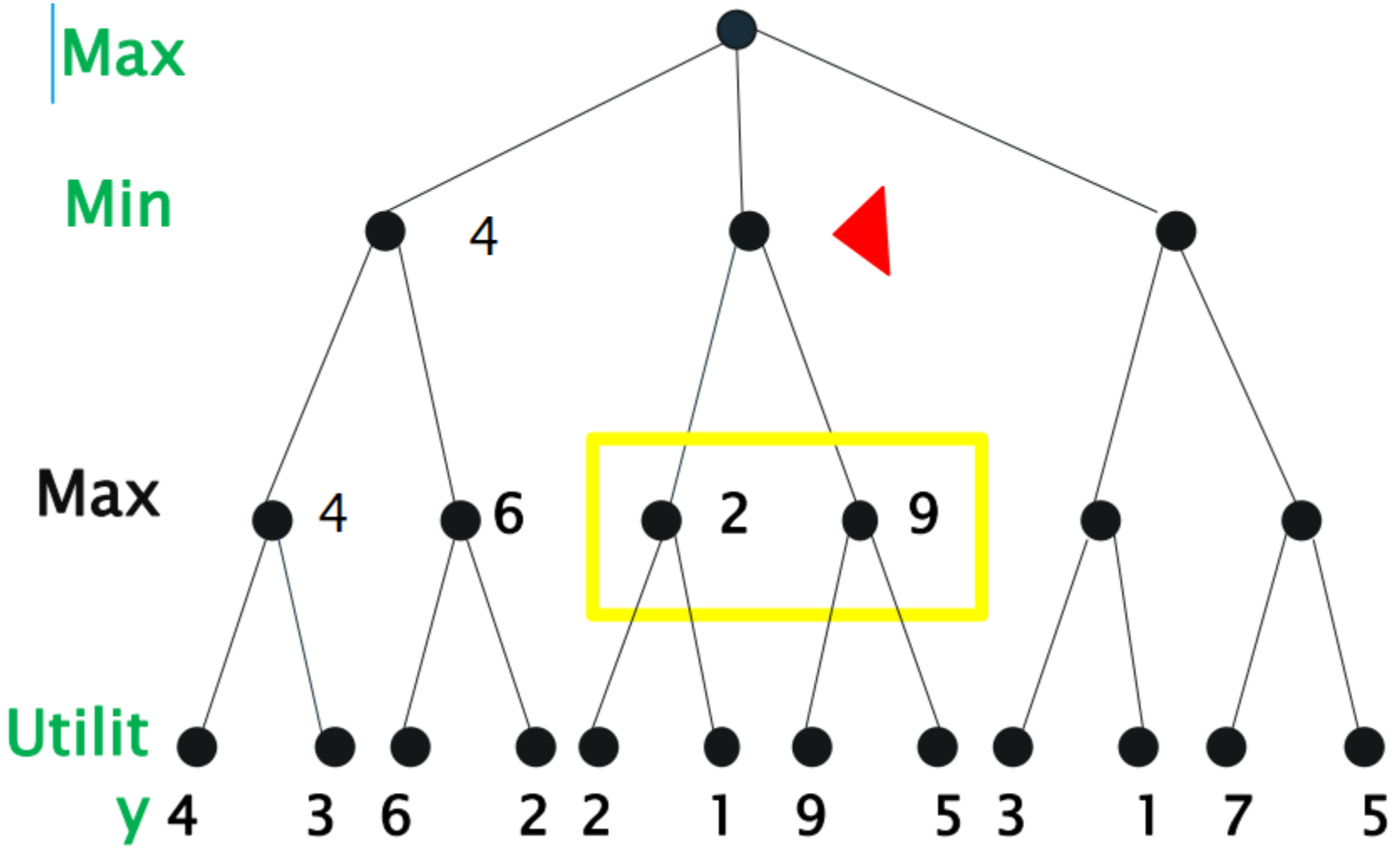
MiniMax Algorithm…



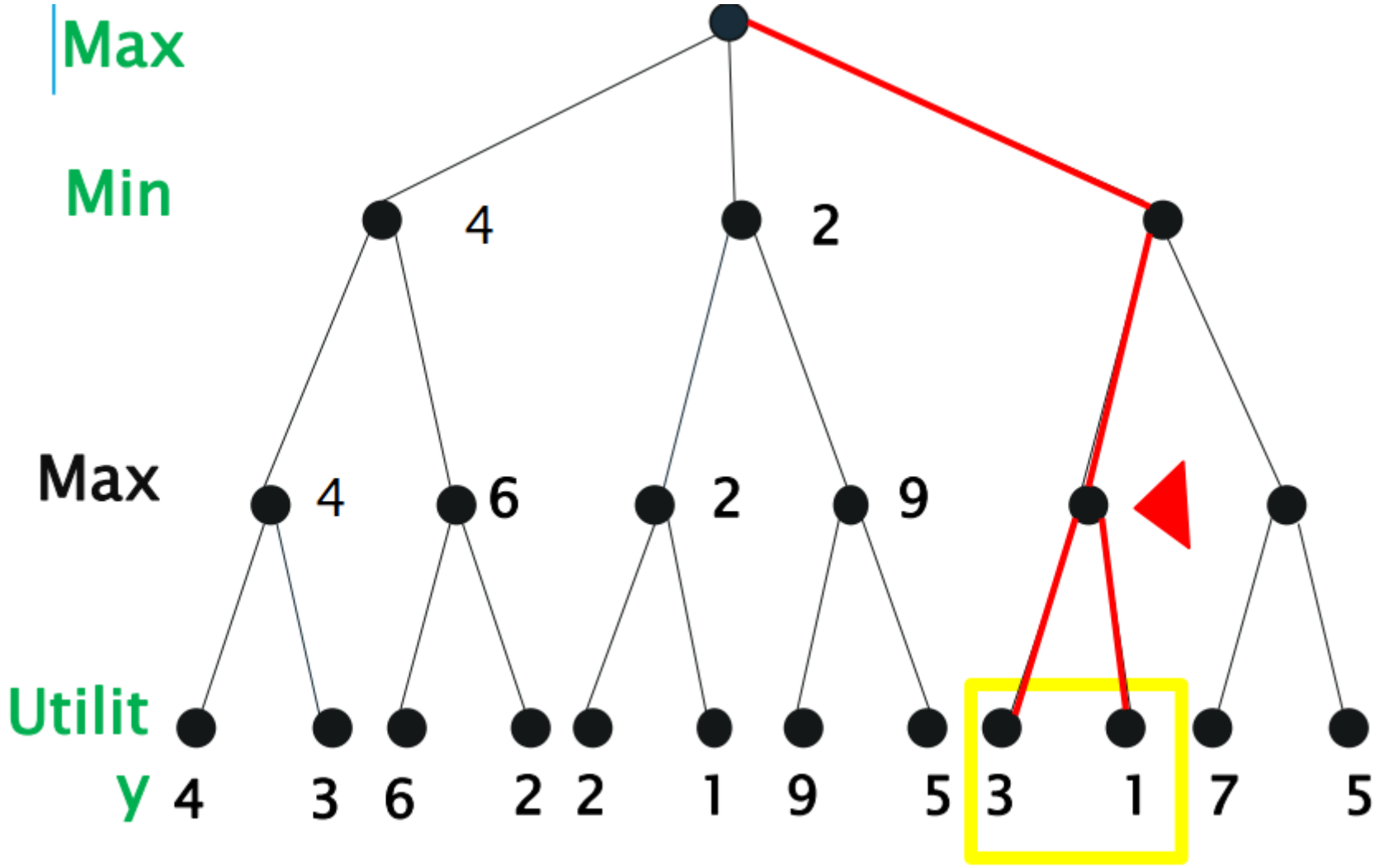
MiniMax Algorithm…



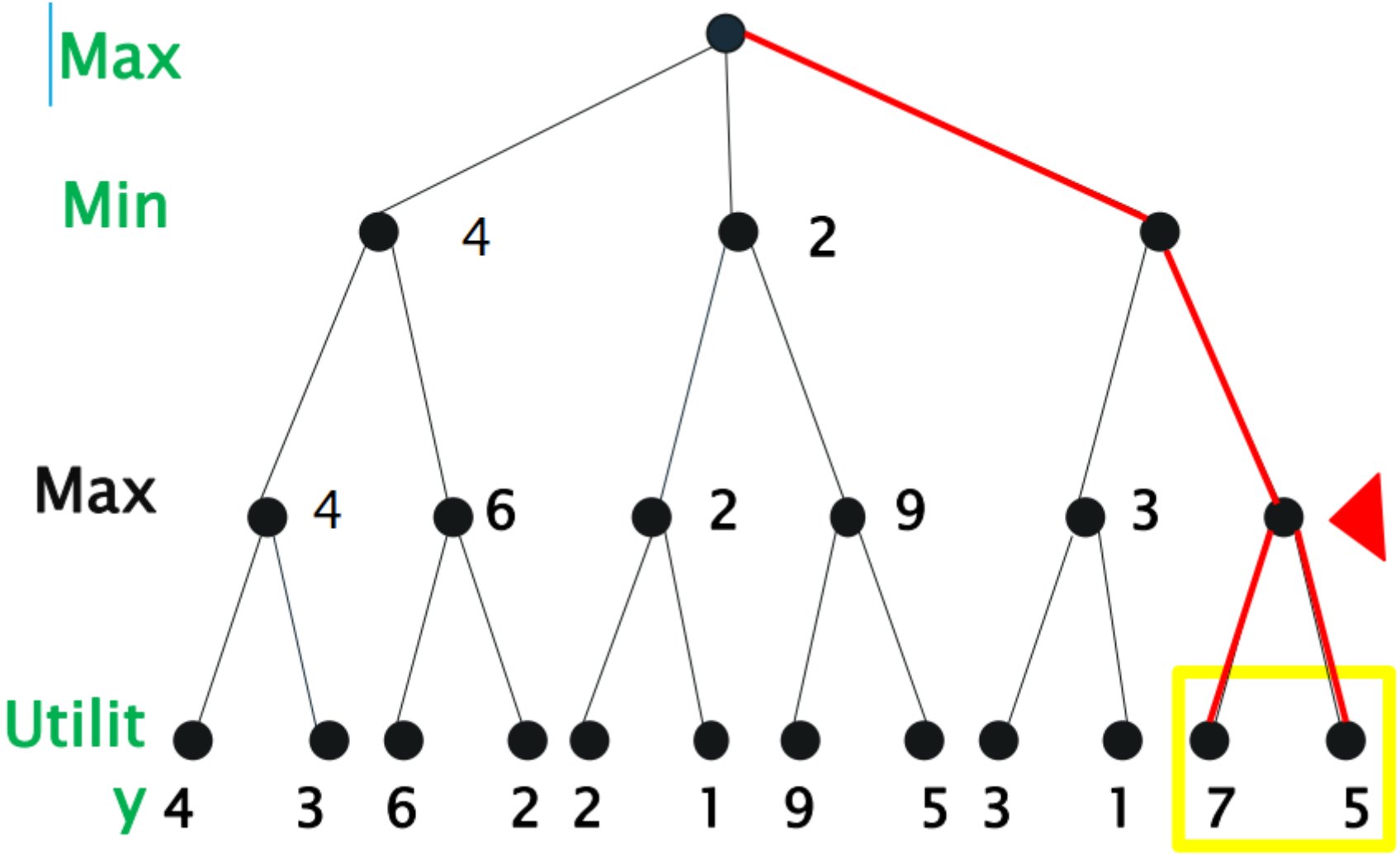
MiniMax Algorithm…



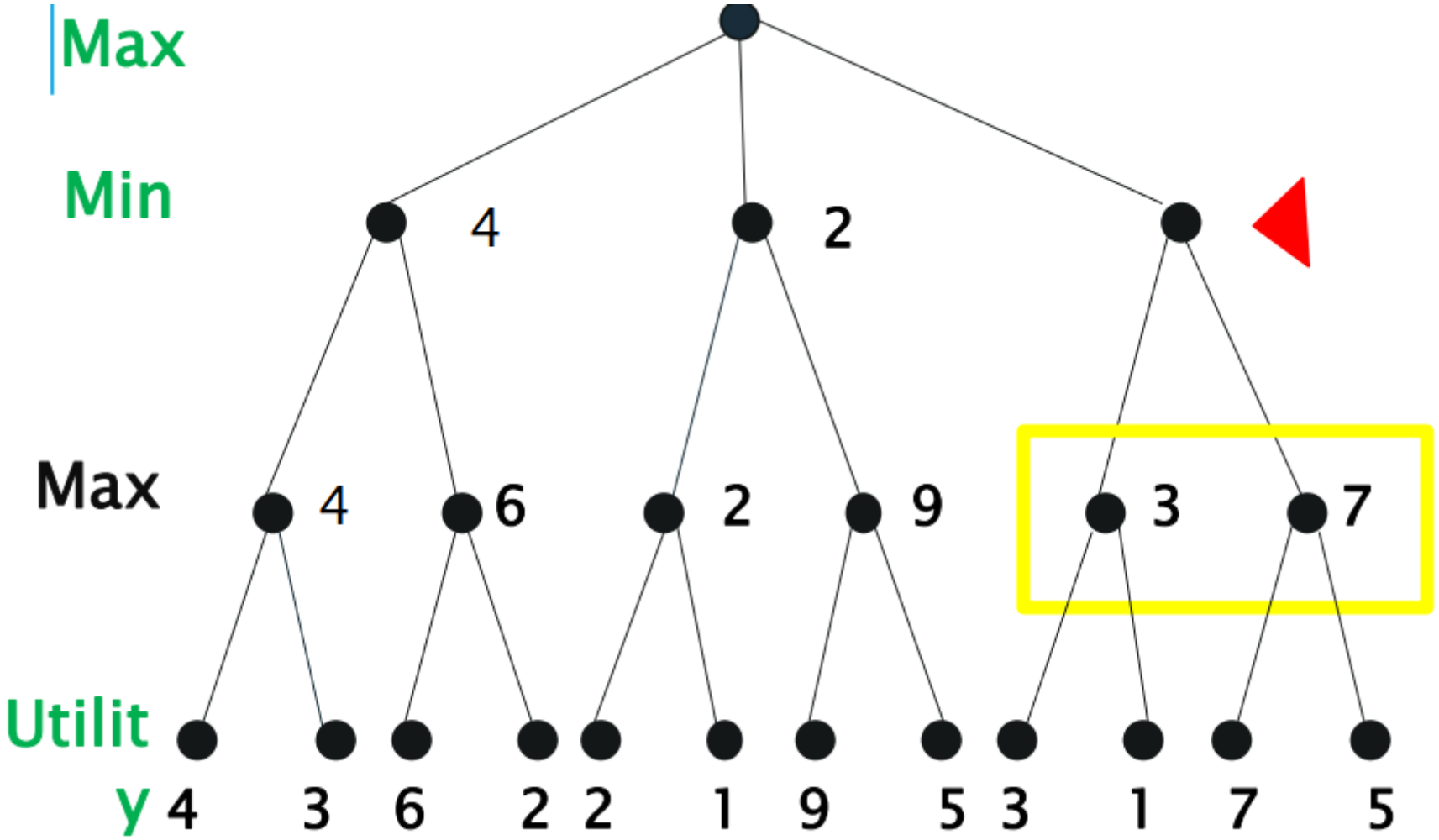
MiniMax Algorithm…



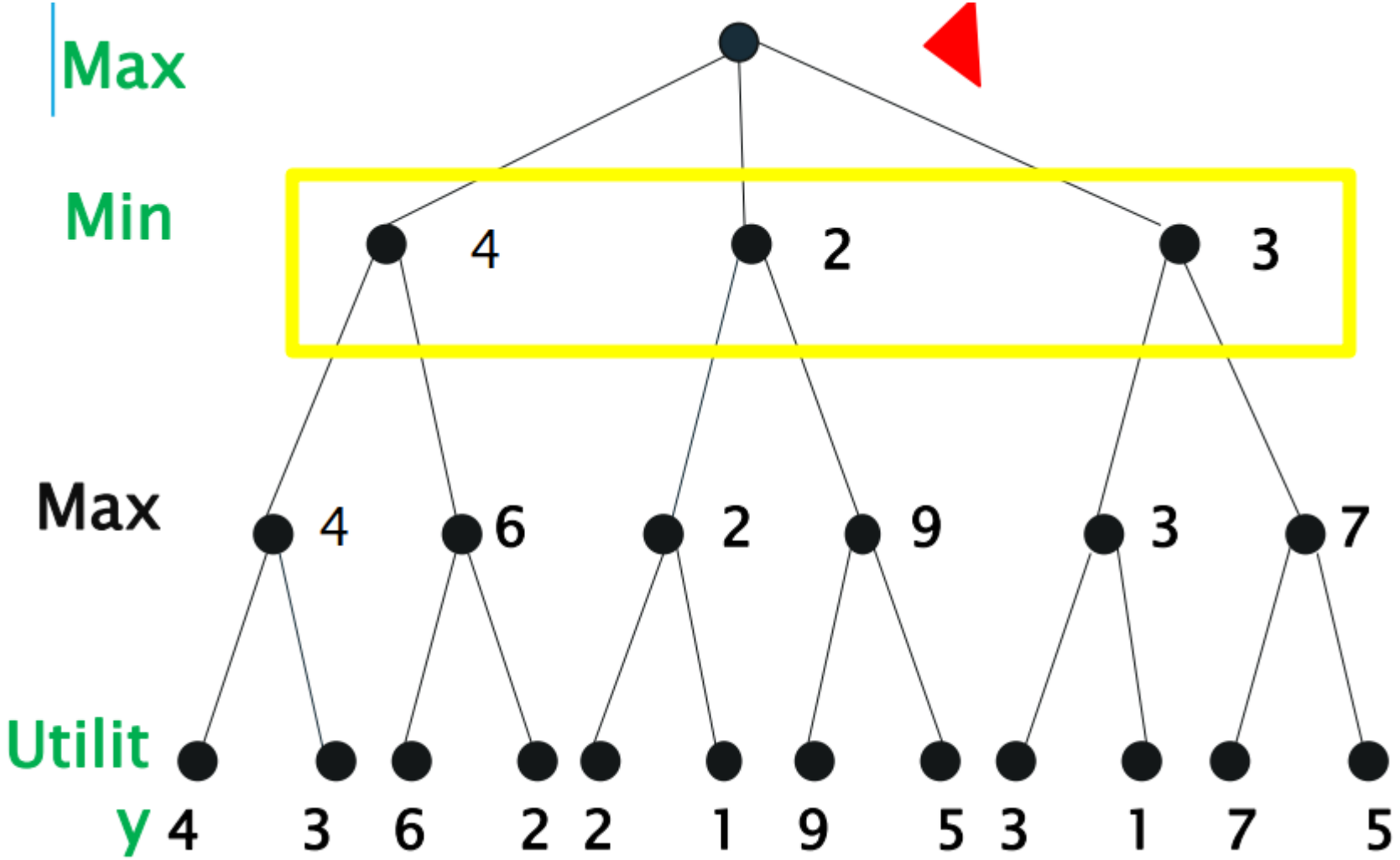
MiniMax Algorithm…



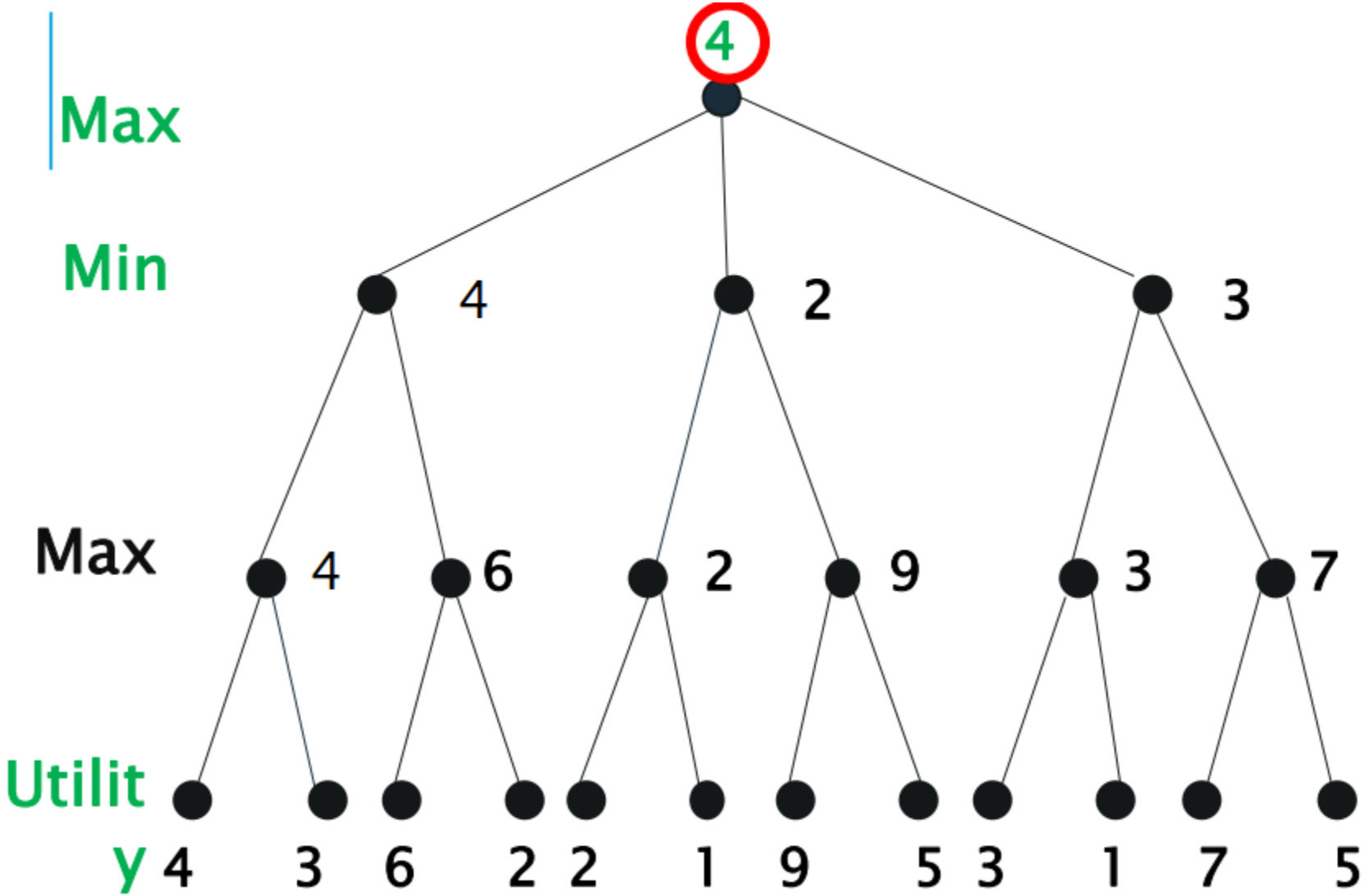
MiniMax Algorithm…



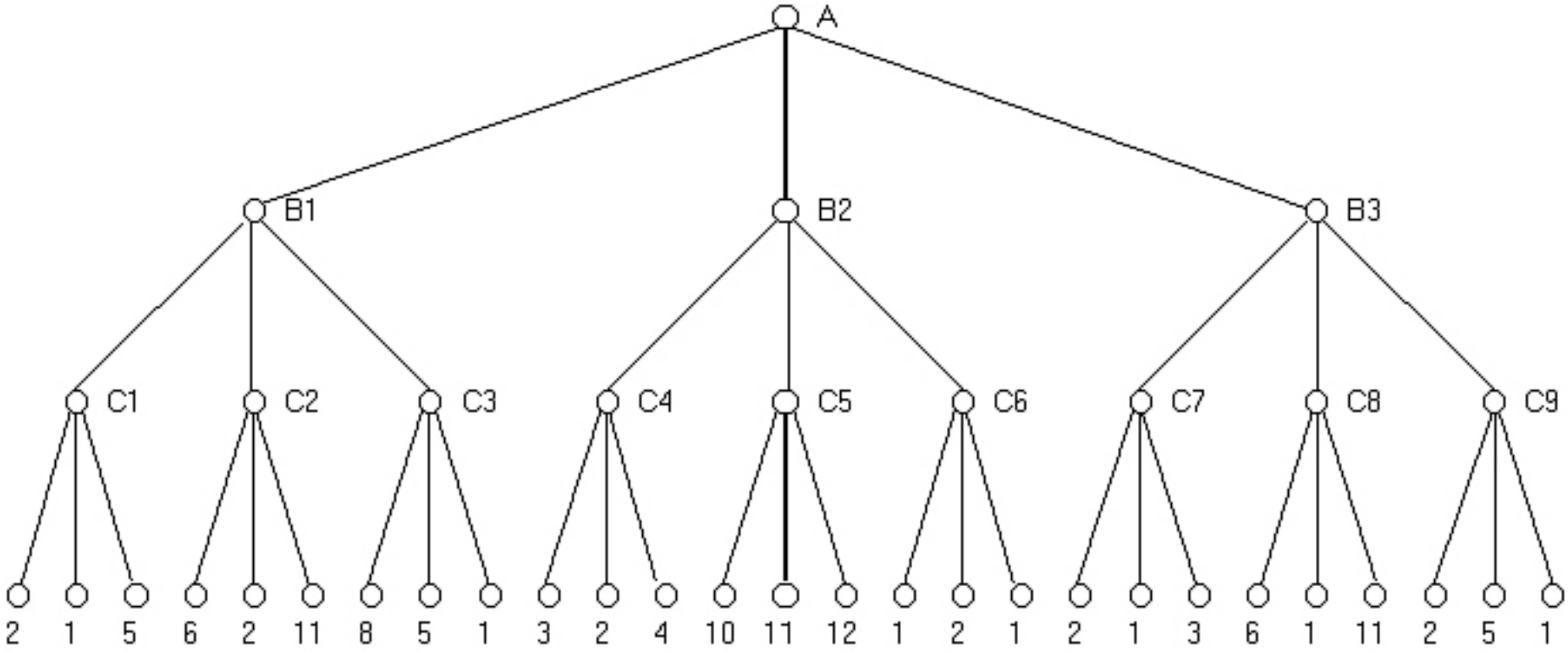
MiniMax Algorithm…



MiniMax Algorithm…



MiniMax Algorithm…



MiniMax Algorithm…

Minimax advantages:

-Returns an optimal action, assuming perfect opponent play.

-Minimax is the simplest possible (reasonable) game search algorithm.

Minimax disadvantages:

-It's completely infeasible in practice. When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut-off states.

α-β Pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

α-β Pruning…

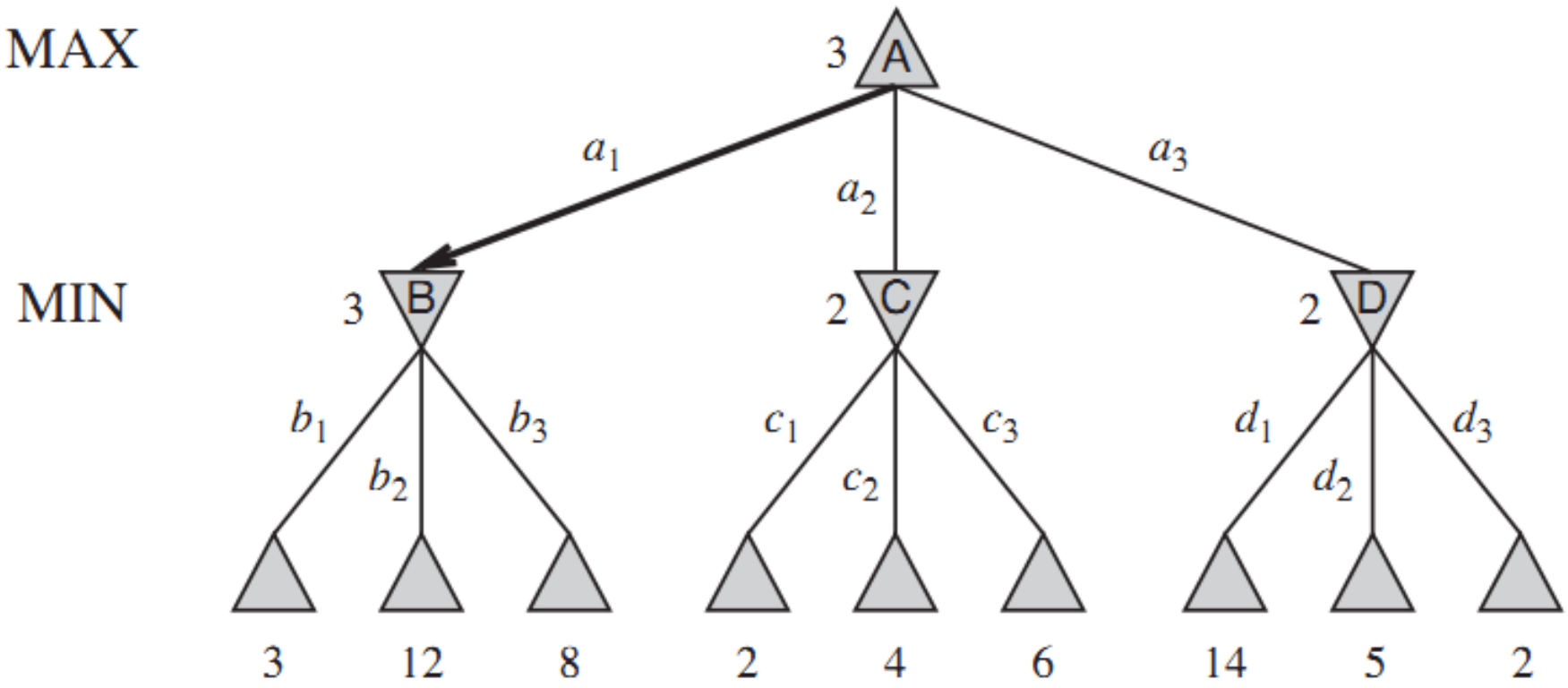
α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

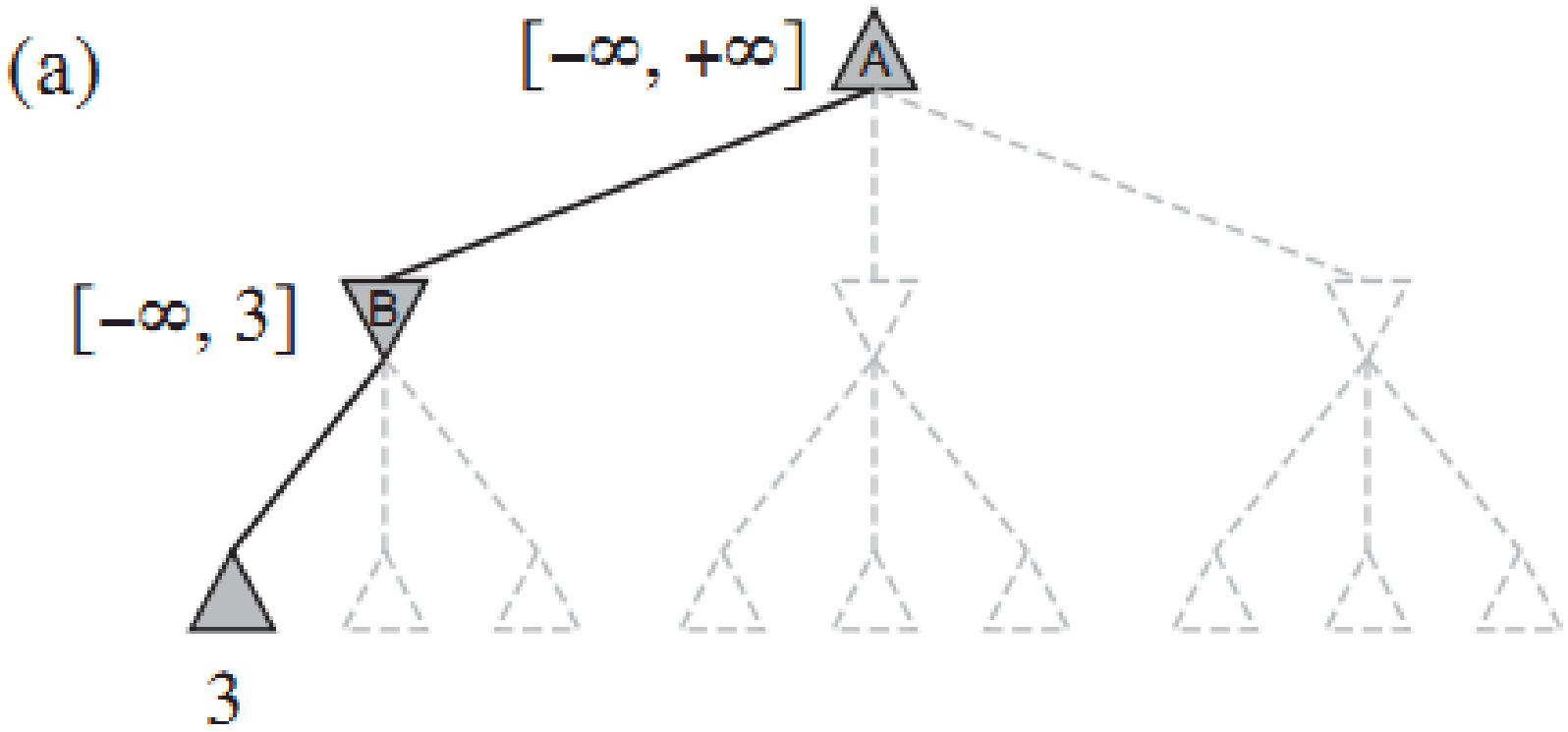
α-β Pruning…

- Values of Alpha and Beta (but not value of V) will be carried out in the downward (i.e. toward the child), but not in the upward (i.e. toward the root). But in the upward, only the value of Alpha or Beta will be changed depending upon the MAX or MIN function respectively, based on the nodes current value V.
- To update the value of Alpha, compare the value of V with value of Beta (i.e. use V >= Beta). If not satisfied, then update, otherwise don't update and prune the rightmost branch.
- To update the value of Beta, compare the value of V with value of Alpha (i.e. use V <= Alpha). If not satisfied, then update, otherwise don't update and prune the rightmost branch.
- Initially, V is –Ve for Alpha, for each node.
- Initially, V is +Ve for Beta, for each node.

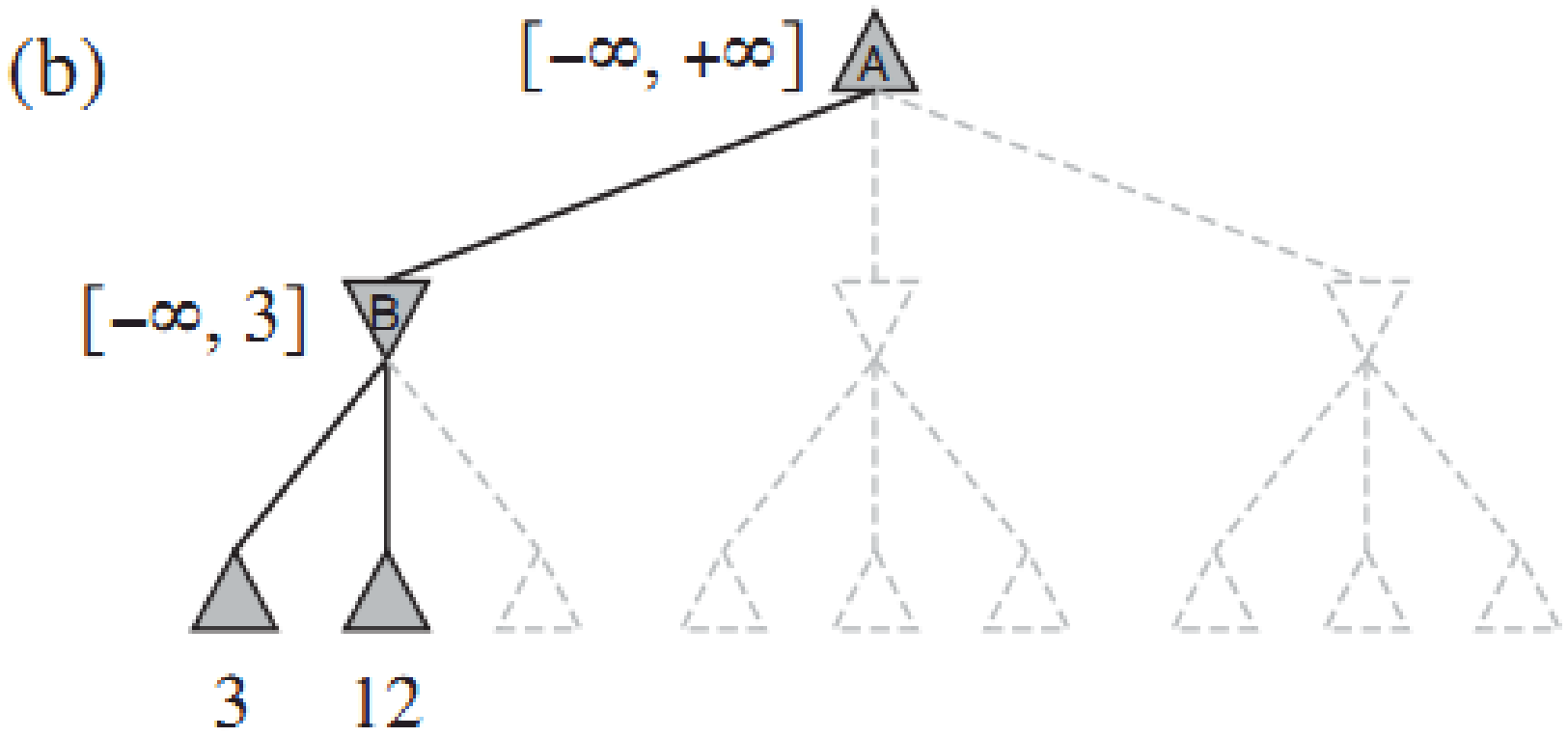
α-β Pruning-Example…



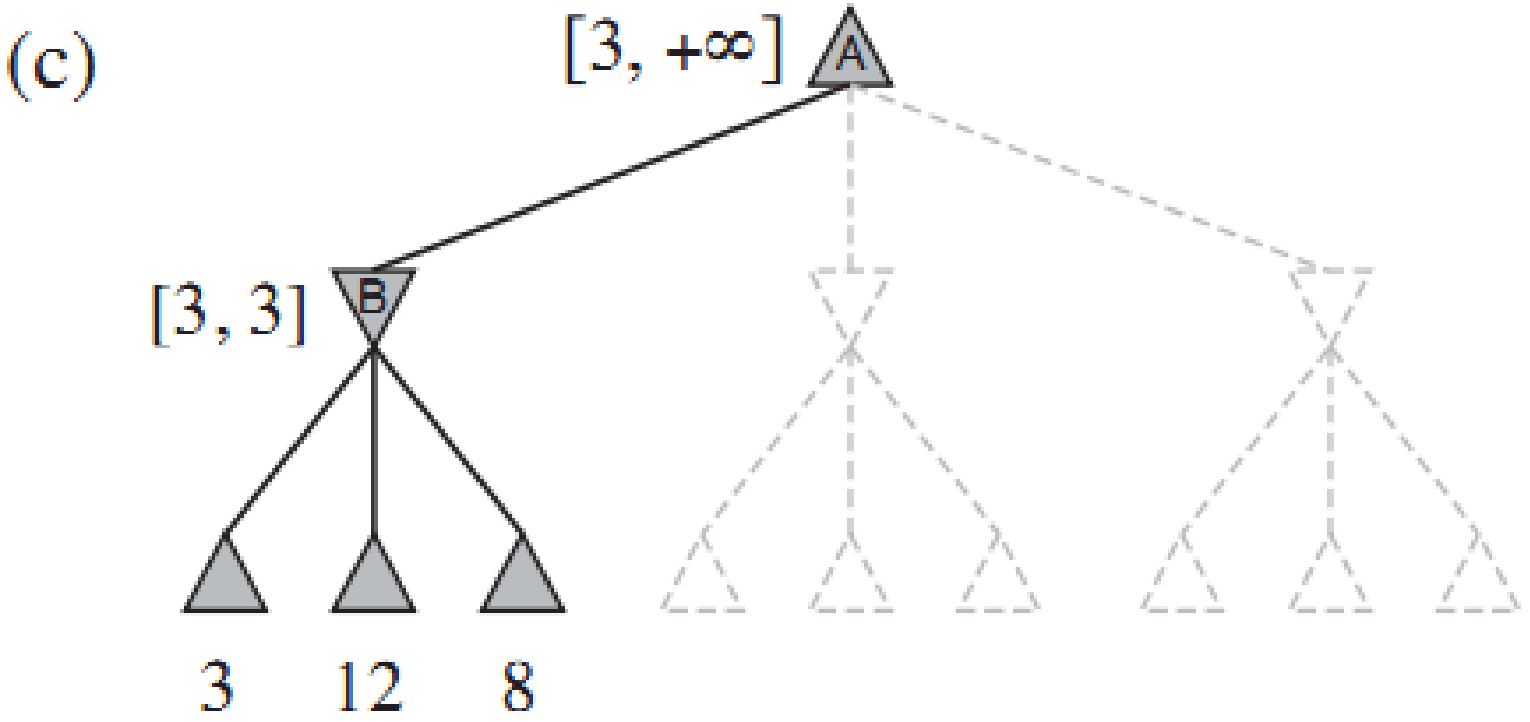
α-β Pruning-Example…



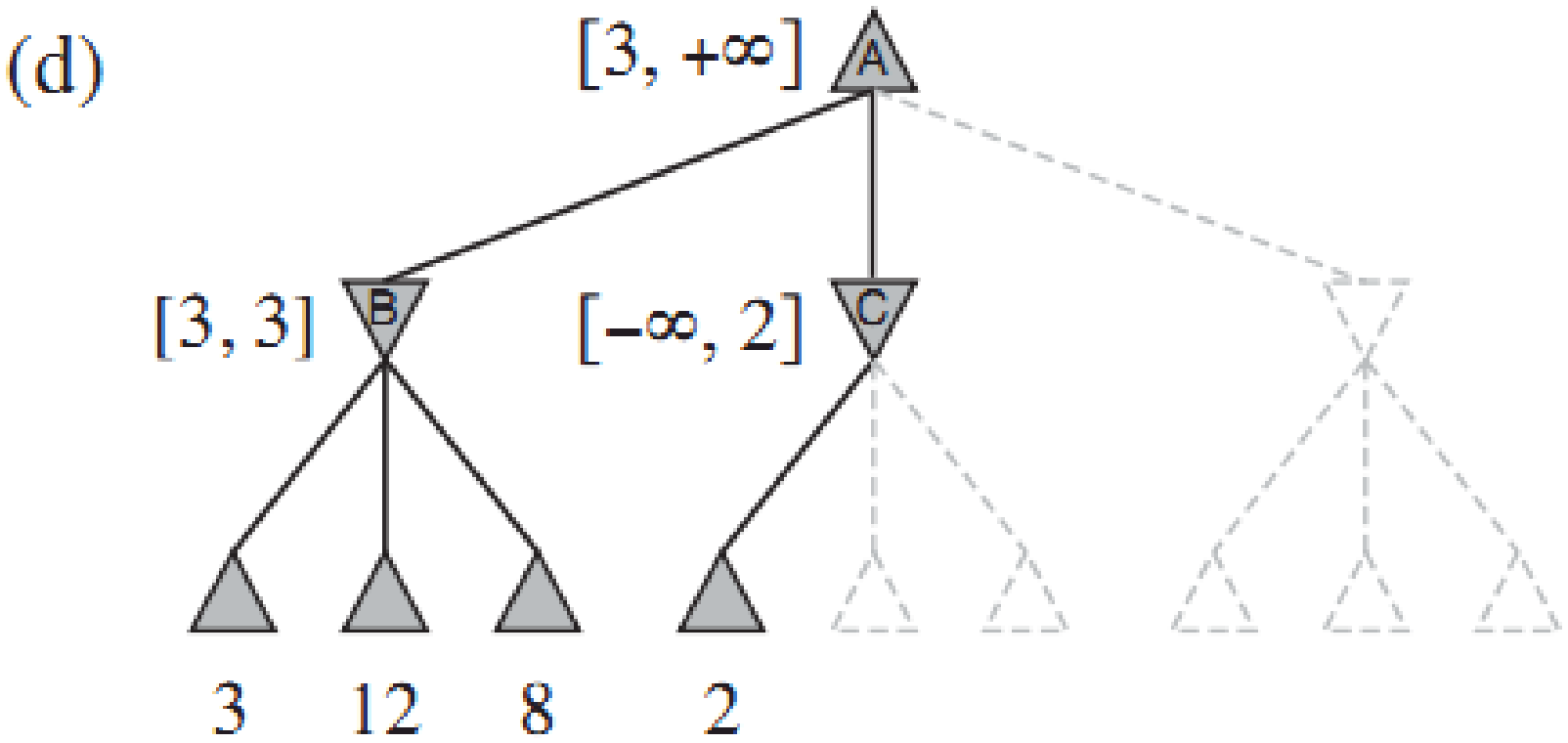
α-β Pruning-Example…



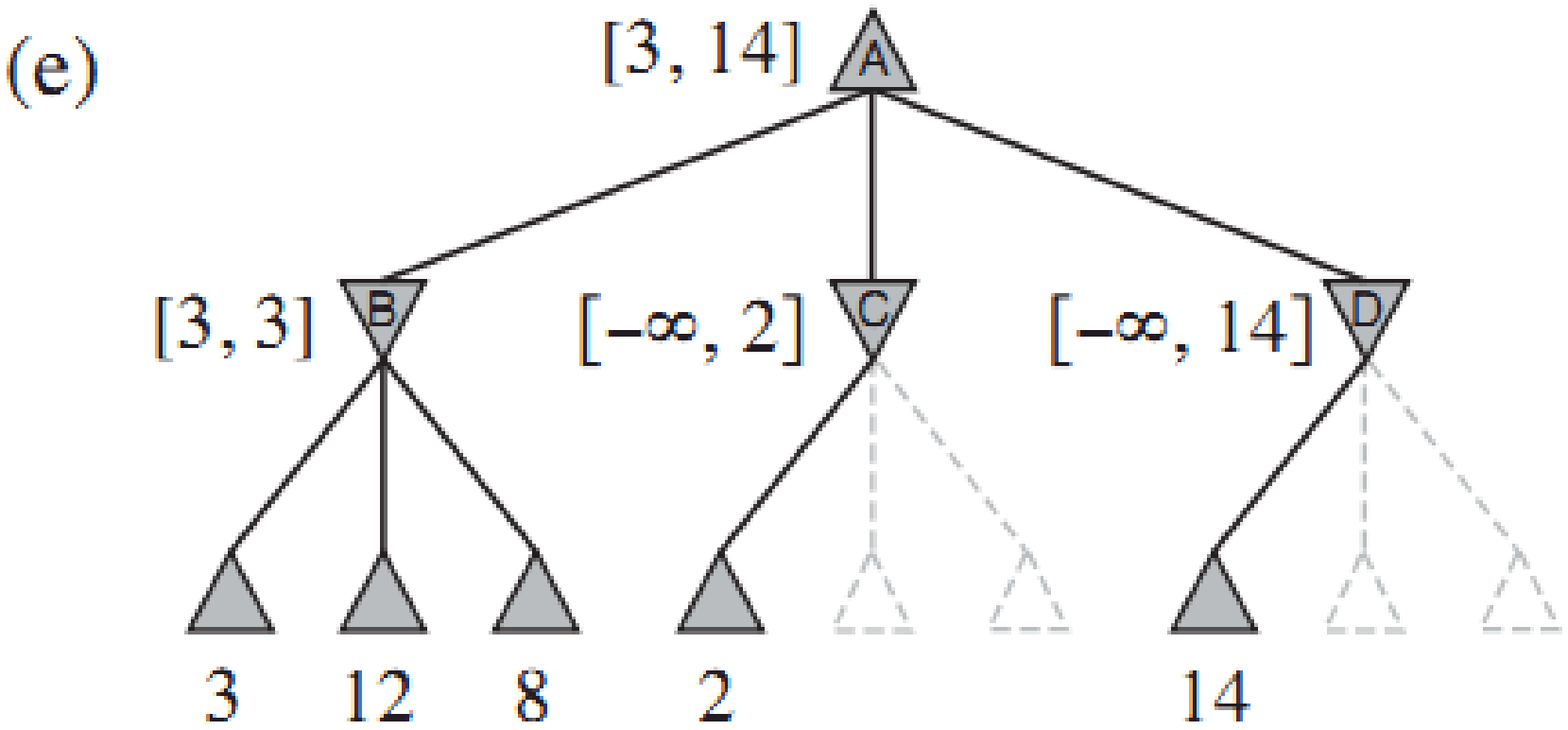
α-β Pruning-Example…



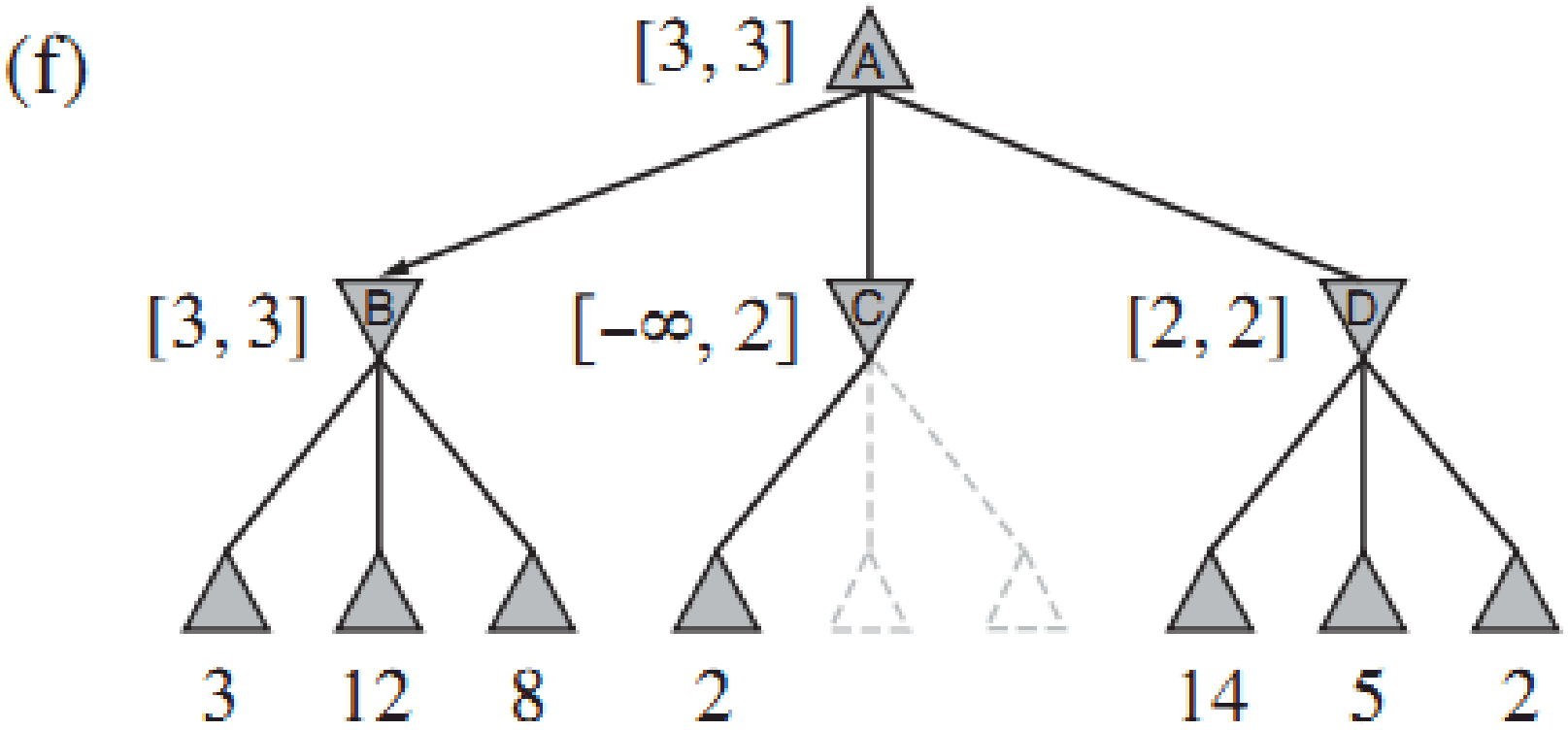
α-β Pruning-Example…



α-β Pruning-Example…

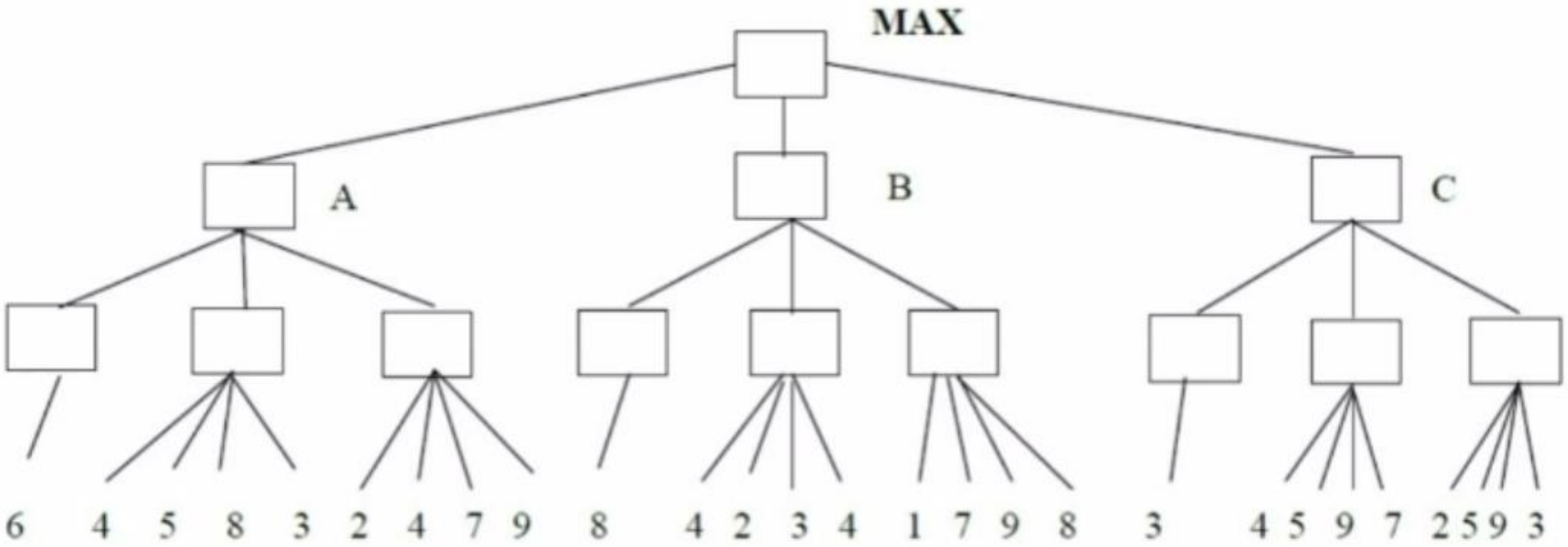


α-β Pruning-Example…



α-β Pruning-Example…

Try yourself



References

[1] http://www.baeldung.com/java-hill-climbing-algorithm
[2]http://www.brainkart.com/article/Hill-Climbing-Search-Algorithm--Concept--AlgorithmAdvantages--Disadvantages_8885/
[3] https://www.geeksforgeeks.org/best-first-search-informed-search/
[4] https://algorithmicthoughts.wordpress.com/2013/01/04/artificial-intelligence-a-search-algorithm/
[5]https://algorithmicthoughts.wordpress.com/2012/12/15/artificial-intelligence-uniform-cost-searchucs/
[6] https://www.seas.upenn.edu/~cis391/Lectures/informed-search-II.pdf
[7] https://www.slideshare.net/foofiM/adversarial-search-45837796
[8] https://www.ics.uci.edu/~welling/teaching/271fall09/UninformedSearch271f09.pdf
[9] http://ai.berkeley.edu/lecture_slides.html

THANKS