Search

https://www.youtube.com/watch?v=KTPmo-KsOis

Search Problem

A set of states (search space)

A initial(start) state

A goal state or goal test(a boolean function which tells us whether a given state is a goal state)

A successor function (a mapping from a state to a set of new states)

Search space (state space)

the feasible region defining the set of all possible solutions

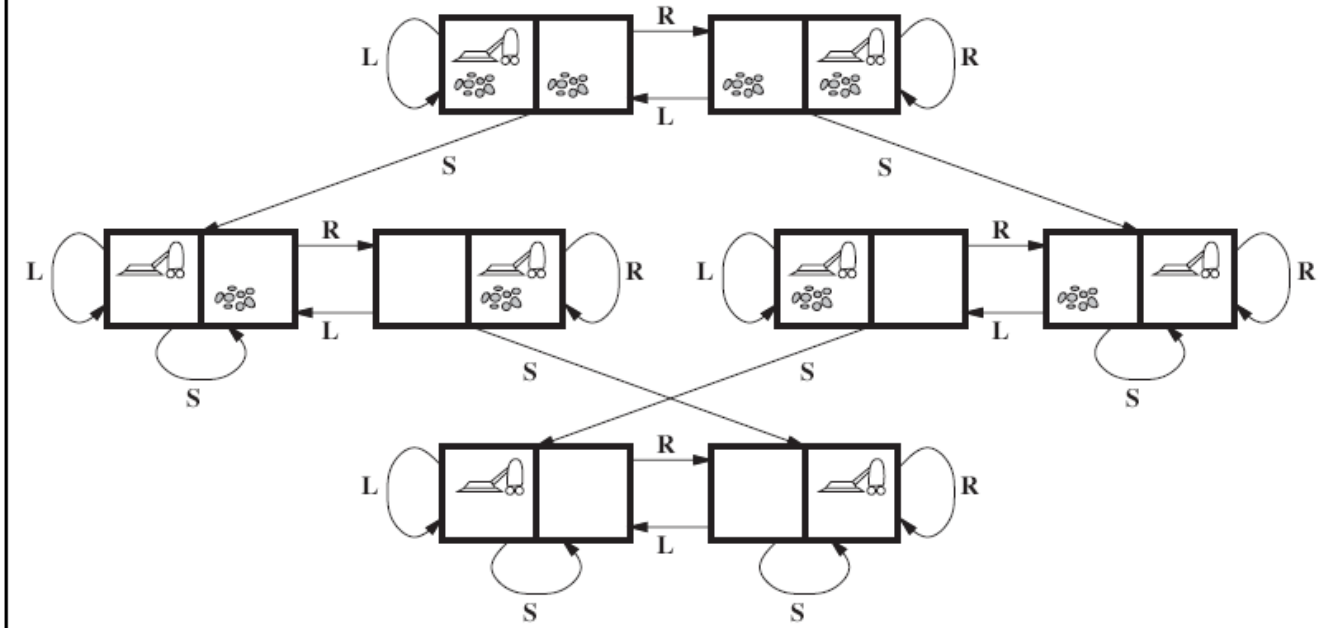


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

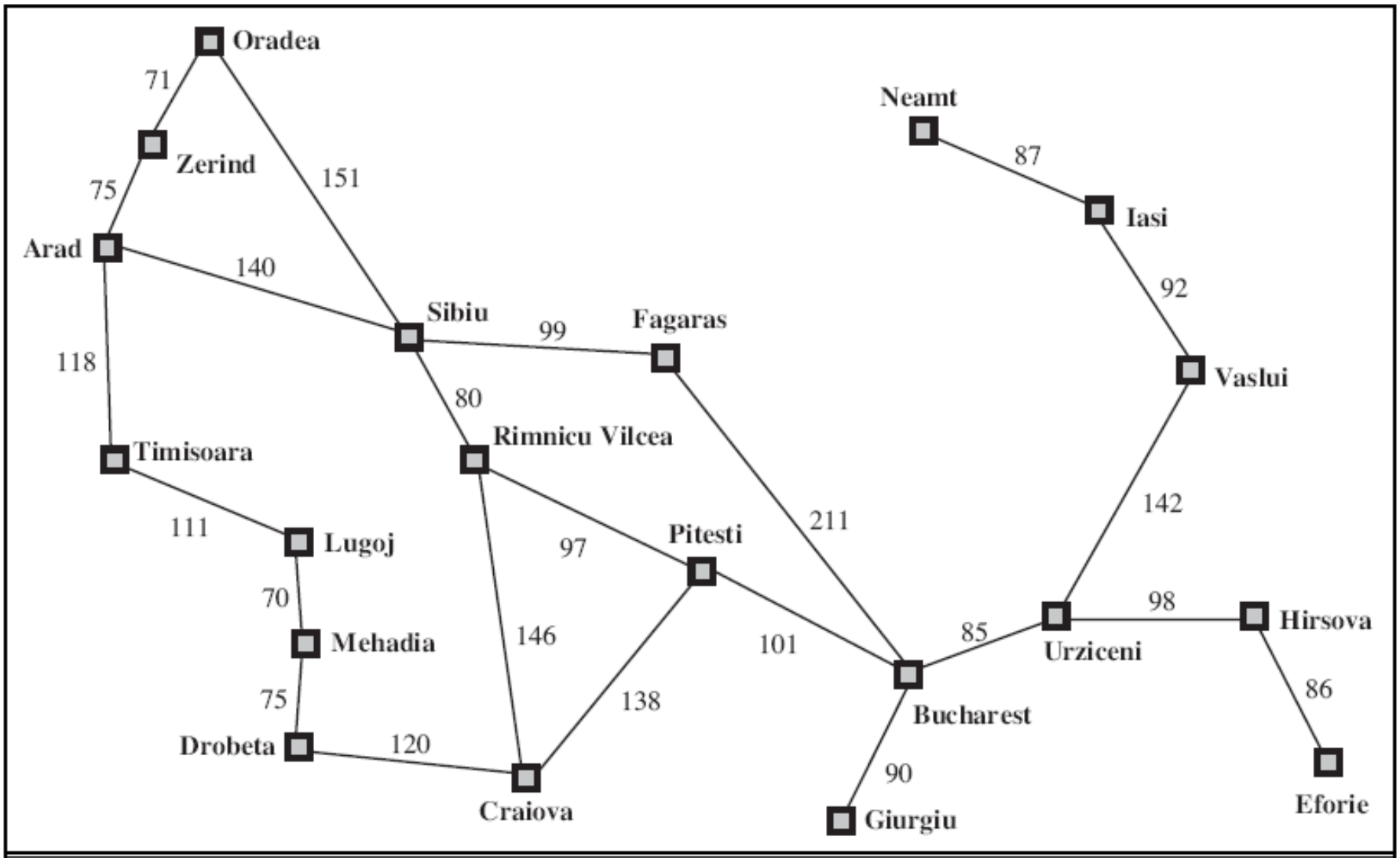


Figure 3.2 A simplified road map of part of Romania.

Search Algorithm

solves the search problem which retrieves information stored within some data structure, or calculated in the search space of a problem domain, either with discrete or continuous values.

Specific applications of search algorithms

Combinatorial optimization

- vehicle routing problem(shortest path problem)
- knapsack problem
- nurse scheduling problem

Constraint satisfaction

- map coloring problem

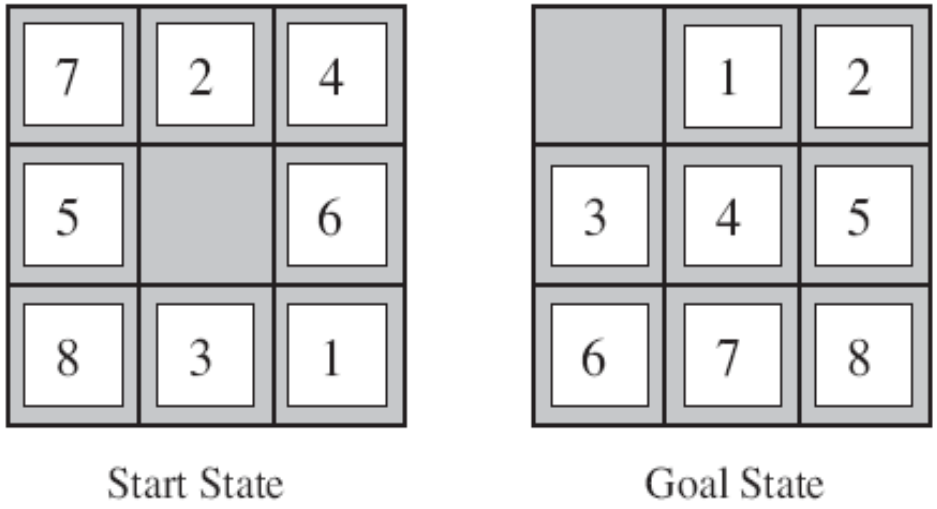
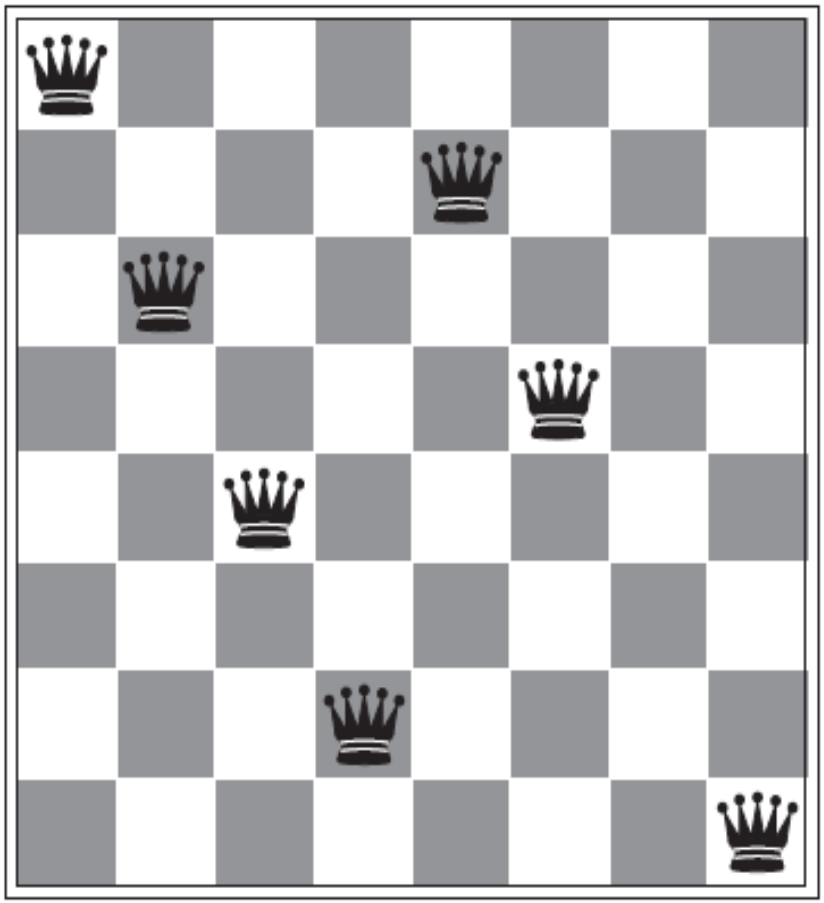


Figure 3.4 A typical instance of the 8-puzzle.

Solving a search problem usually involves starting in some initial state and trying to arrive at a specified goal state or any state in a set of goal states. The actions are selected in a way that tries to make this happen.



- Place eight chess queens on an 8×8 chessboard so that no two queens threaten each other.
- No two queens share the same row, column, or diagonal.

Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

Home work assignment:

Solve for 9-Queens Problem and 10-Queens Problem. Draw a solution for each.

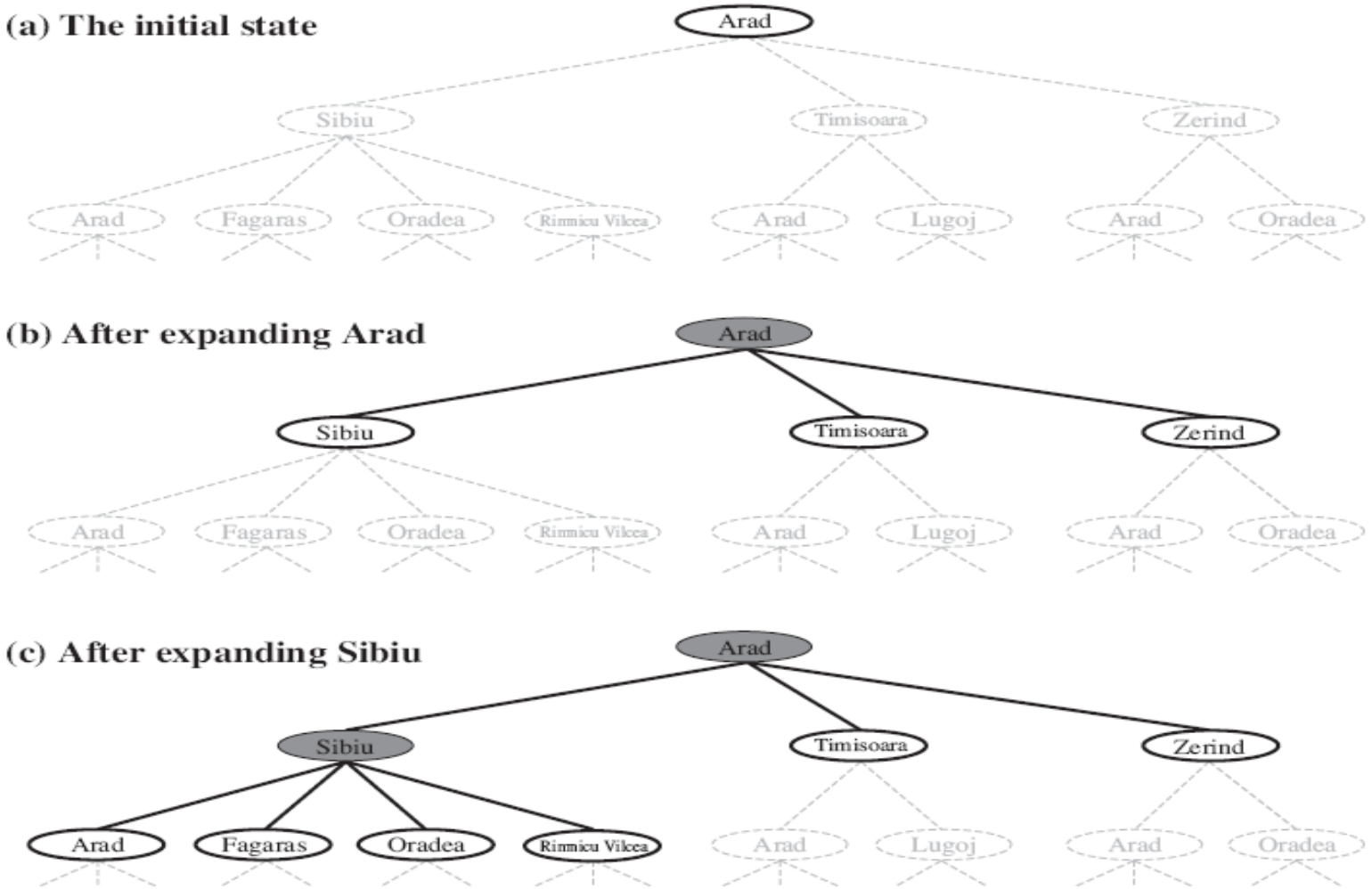


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

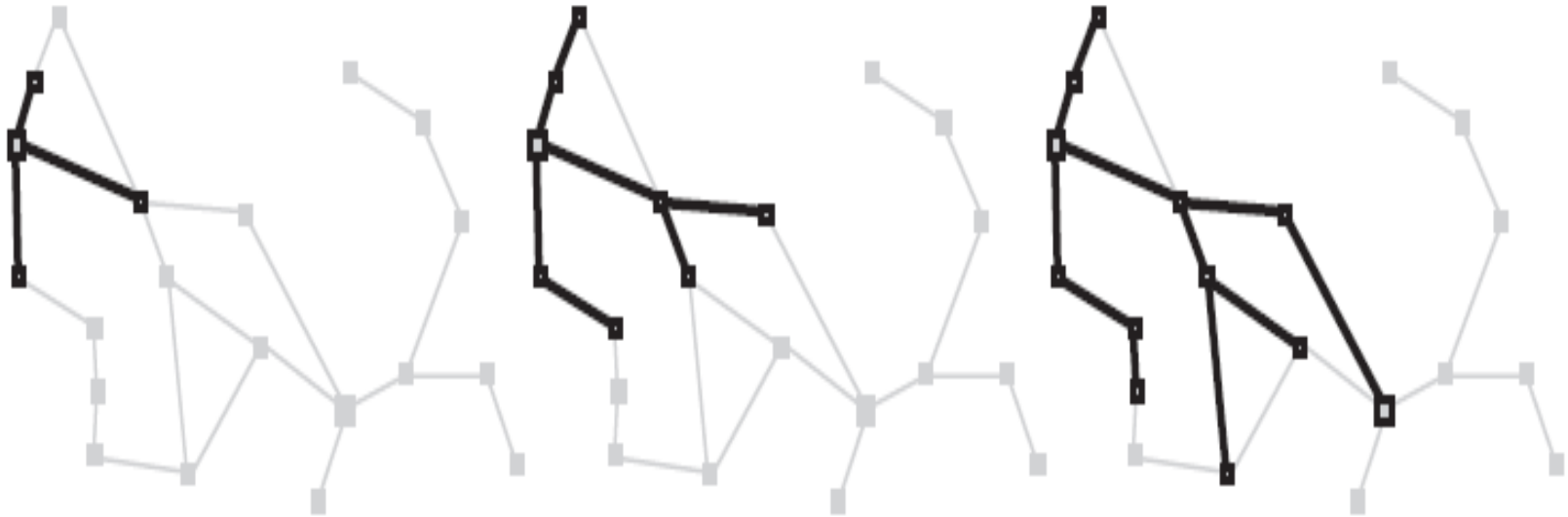
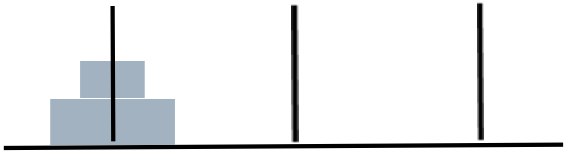


Figure 3.8 A sequence of search trees generated by a graph search on the Romania problem of Figure 3.2. At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

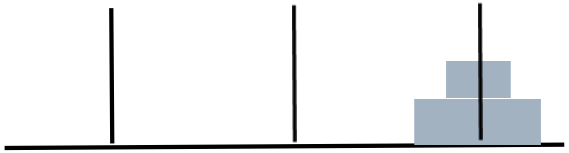
Class practice problem:

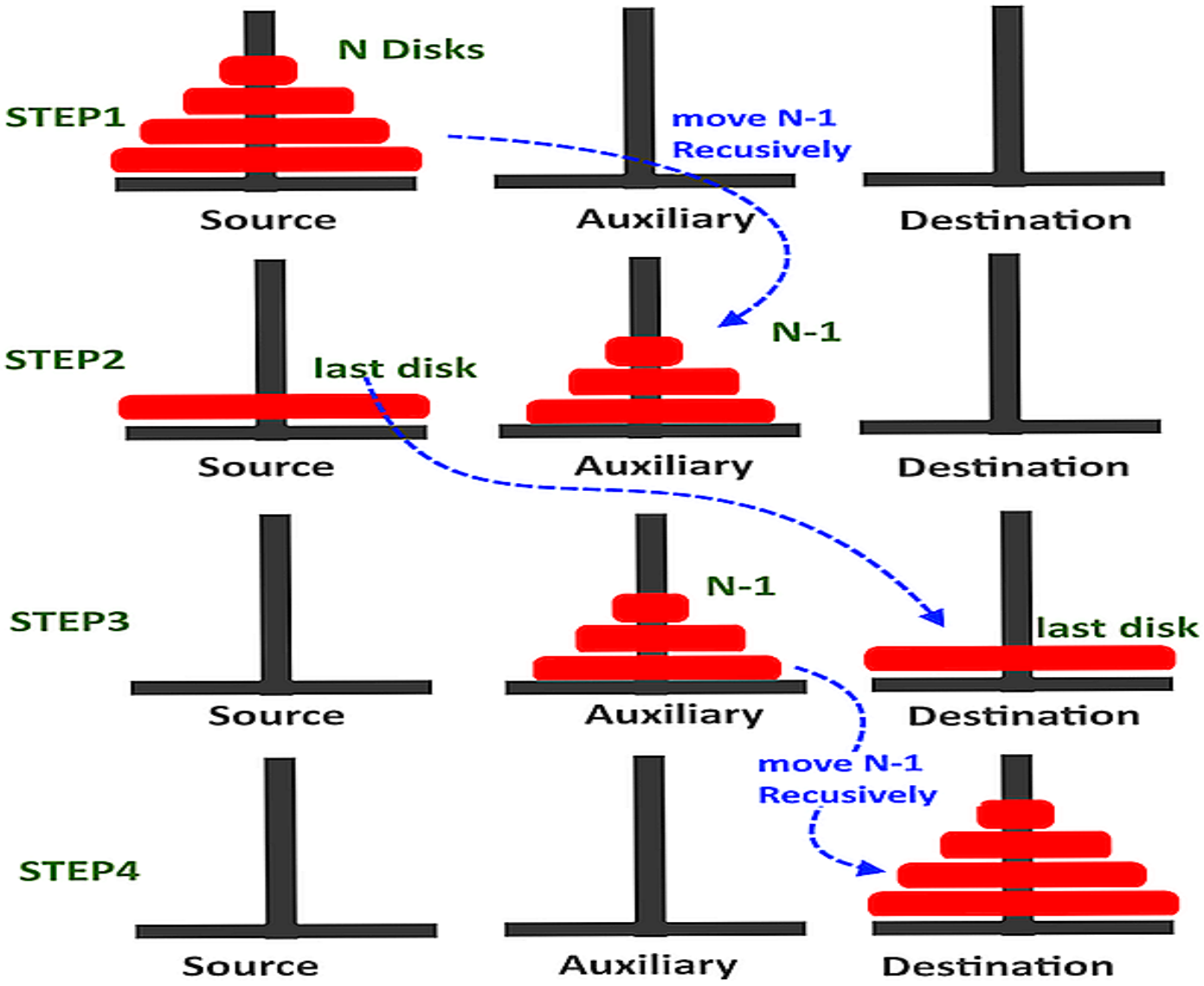
Draw the search tree of a Hanoi Tower with 2 disks and identify the solution path.

Initial state:



Goal state:




N Disks
STEP1
Source
Auxiliary
Destination
move N-1
Recusively
N-1
STEP2
last disk
Source
Auxiliary
Destination
STEP3
N-1
last disk
Source
Auxiliary
Destination
move N-1
Recusively
STEP4
Source
Auxiliary
Destination

a recursive solution for the Towers of Hanoi puzzle with 4 disks

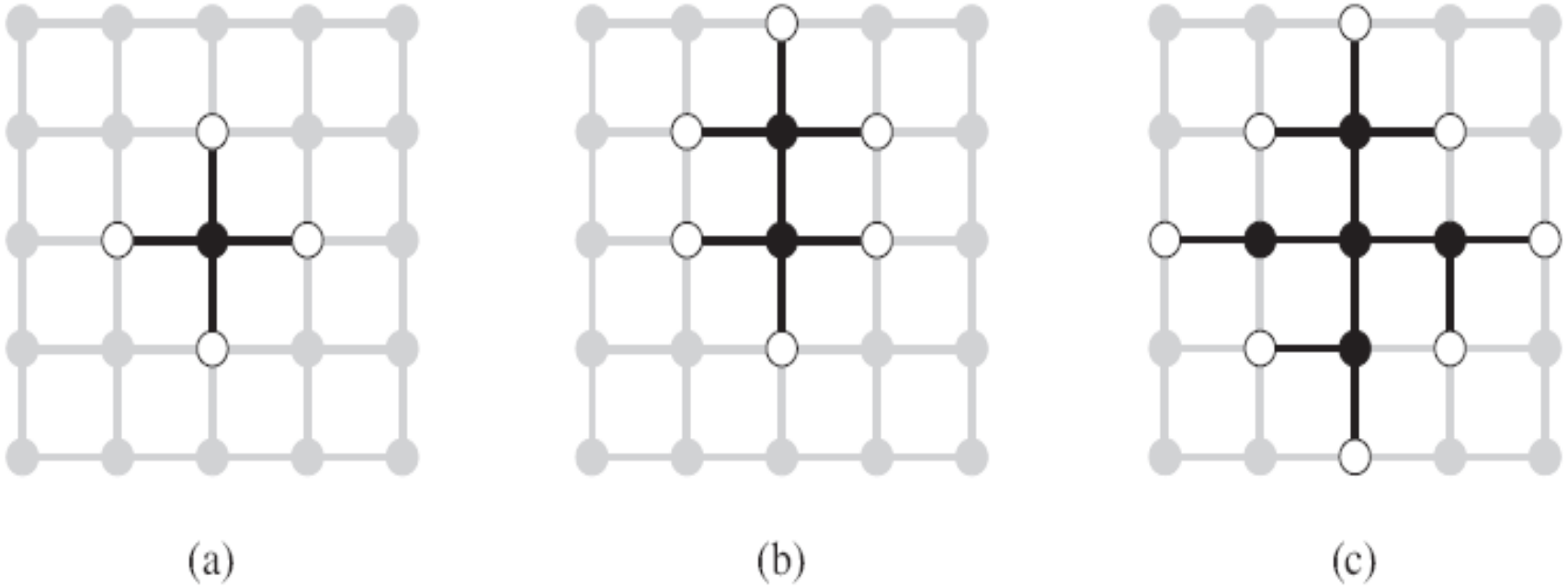


Figure 3.9 The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

UninformedSearch(Blind Search)

https://youtu.be/5OJv6iHMtVw

- Breadth-First Search
- Depth-First Search
- Uniform Cost Search
- Depth-Limited Search
- Iterative Deepening Depth-First Search
- Bidirectional Search

Breadth-first search explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level

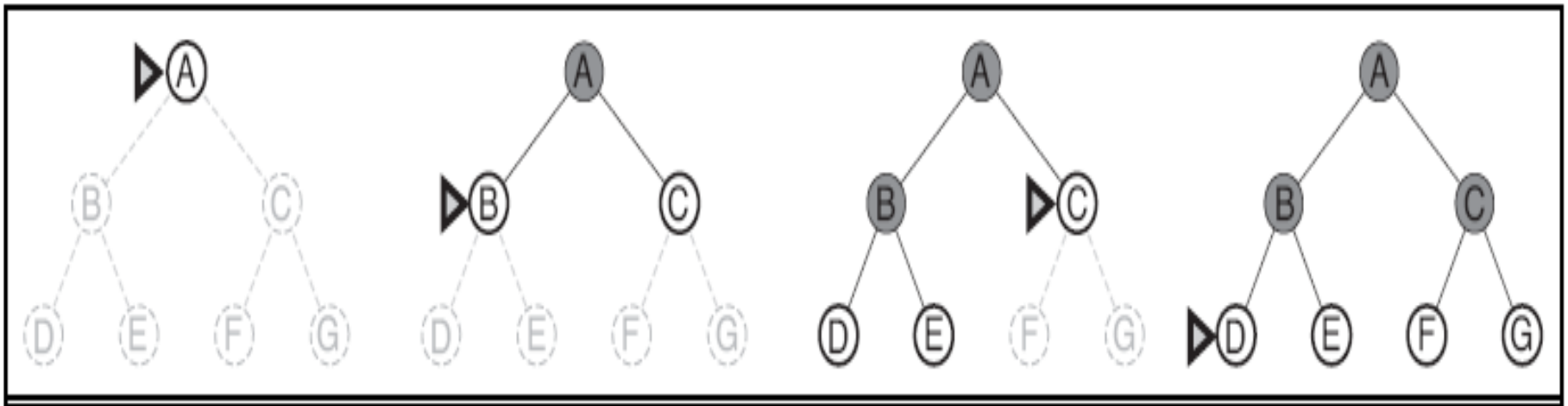


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

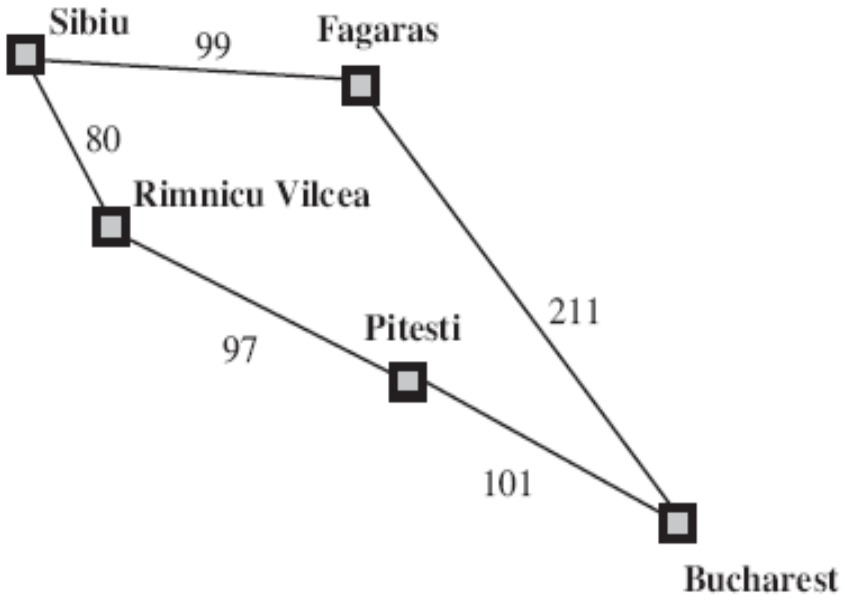


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

Uniform Cost Search
https://www.youtube.com/watch?v=dRMvK76xQJI

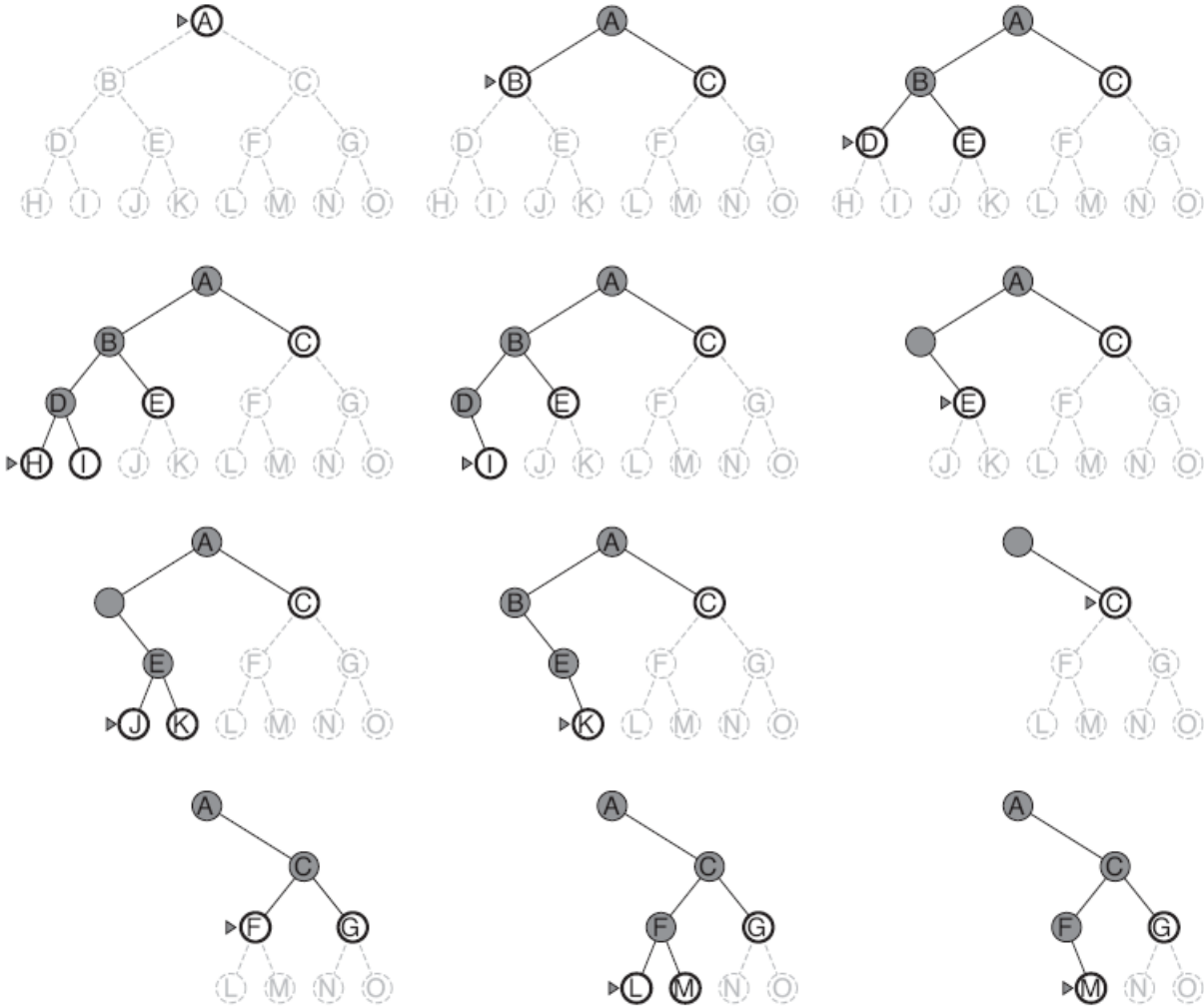


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Depth-first search explores as far as possible along each branch before backtracking.

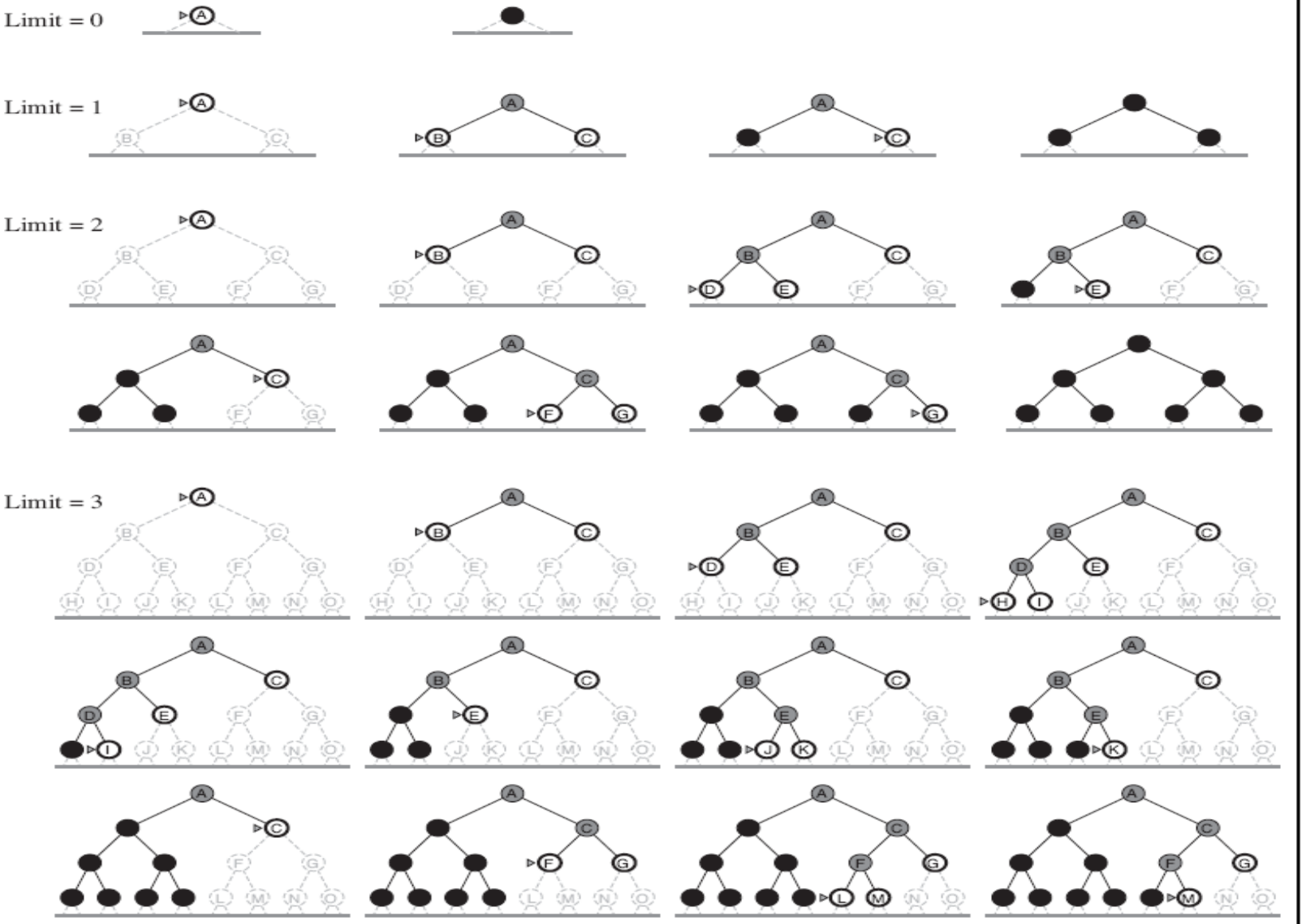


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

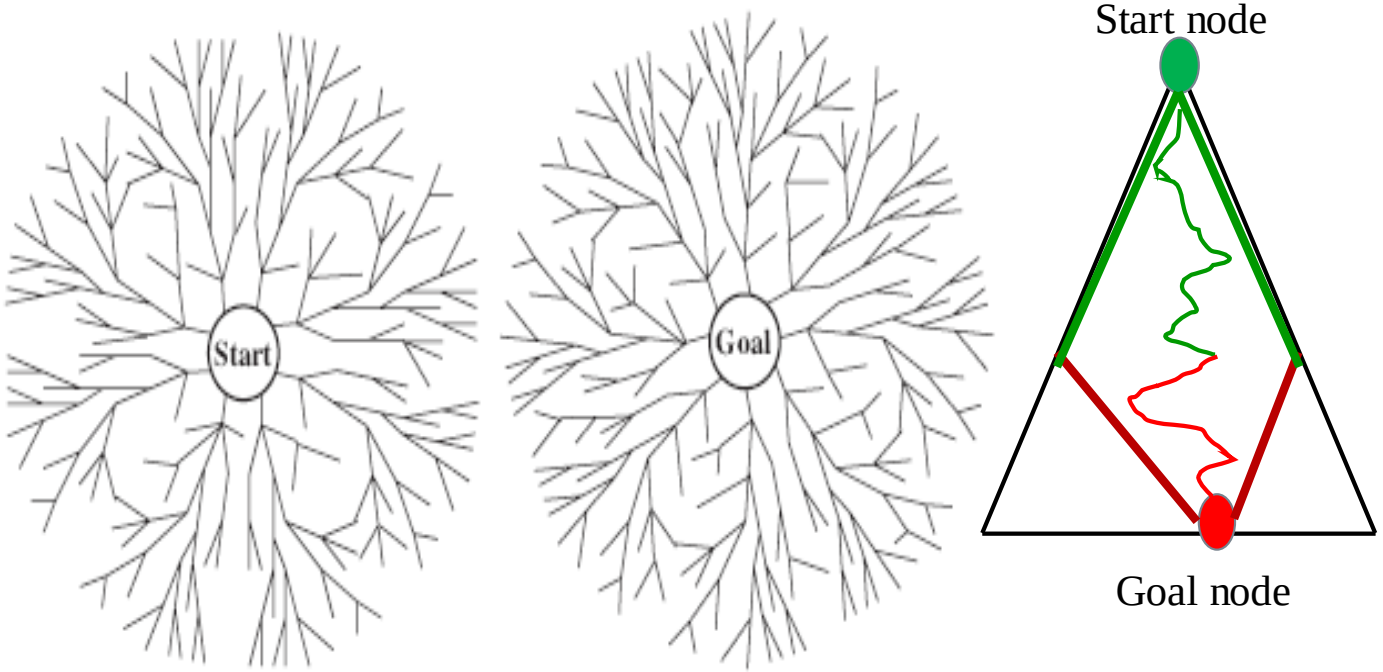


Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

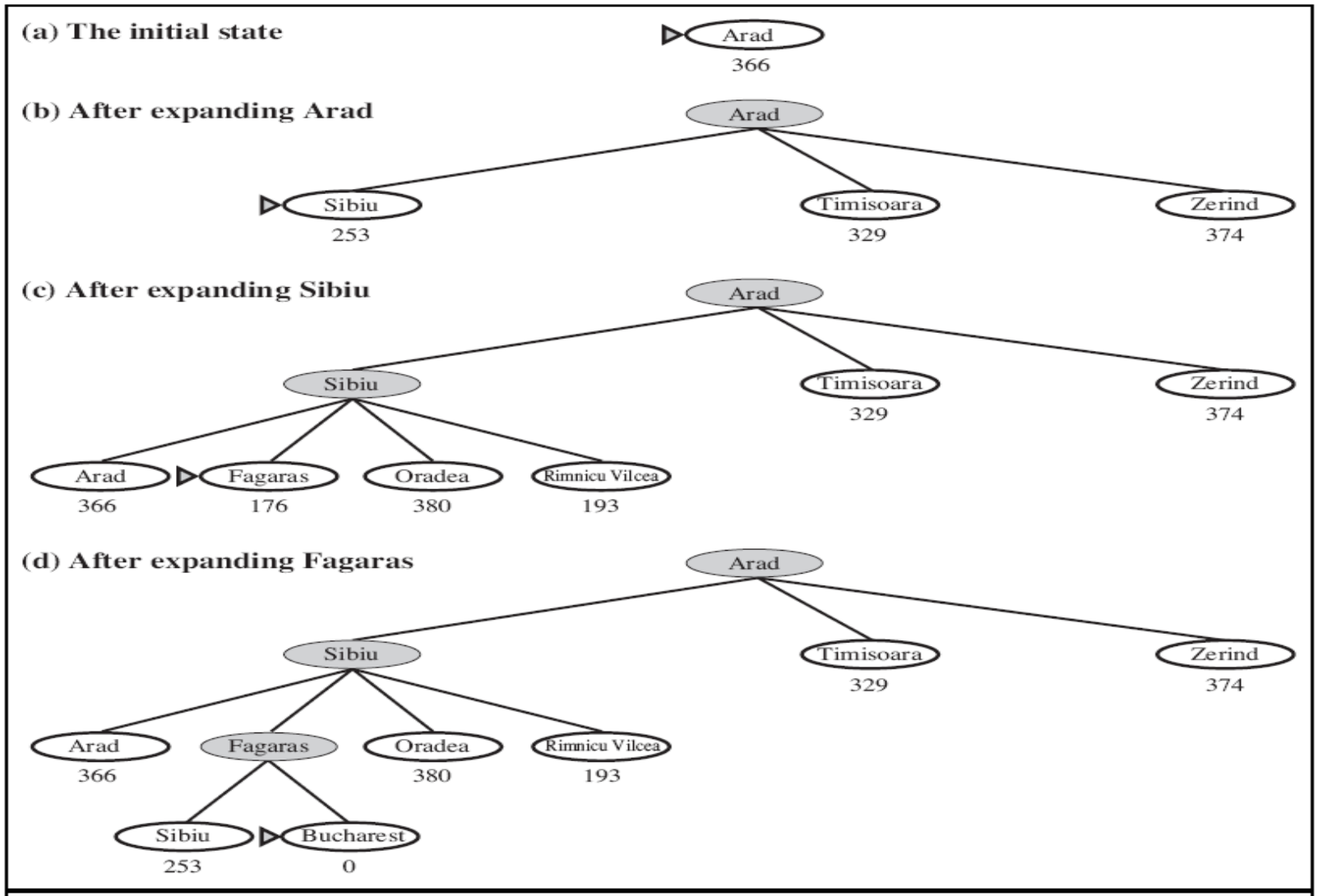


Figure 3.23 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

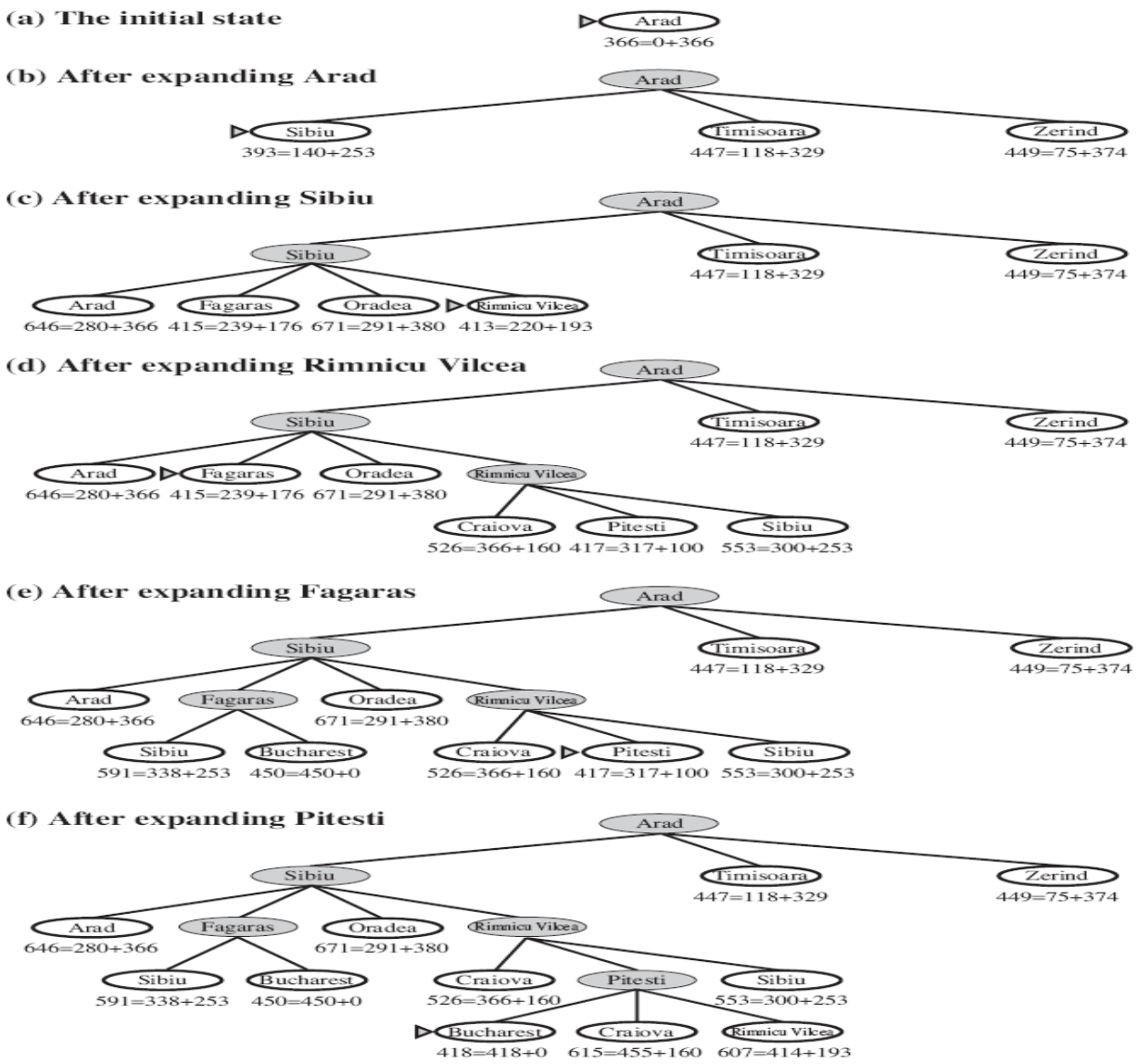


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

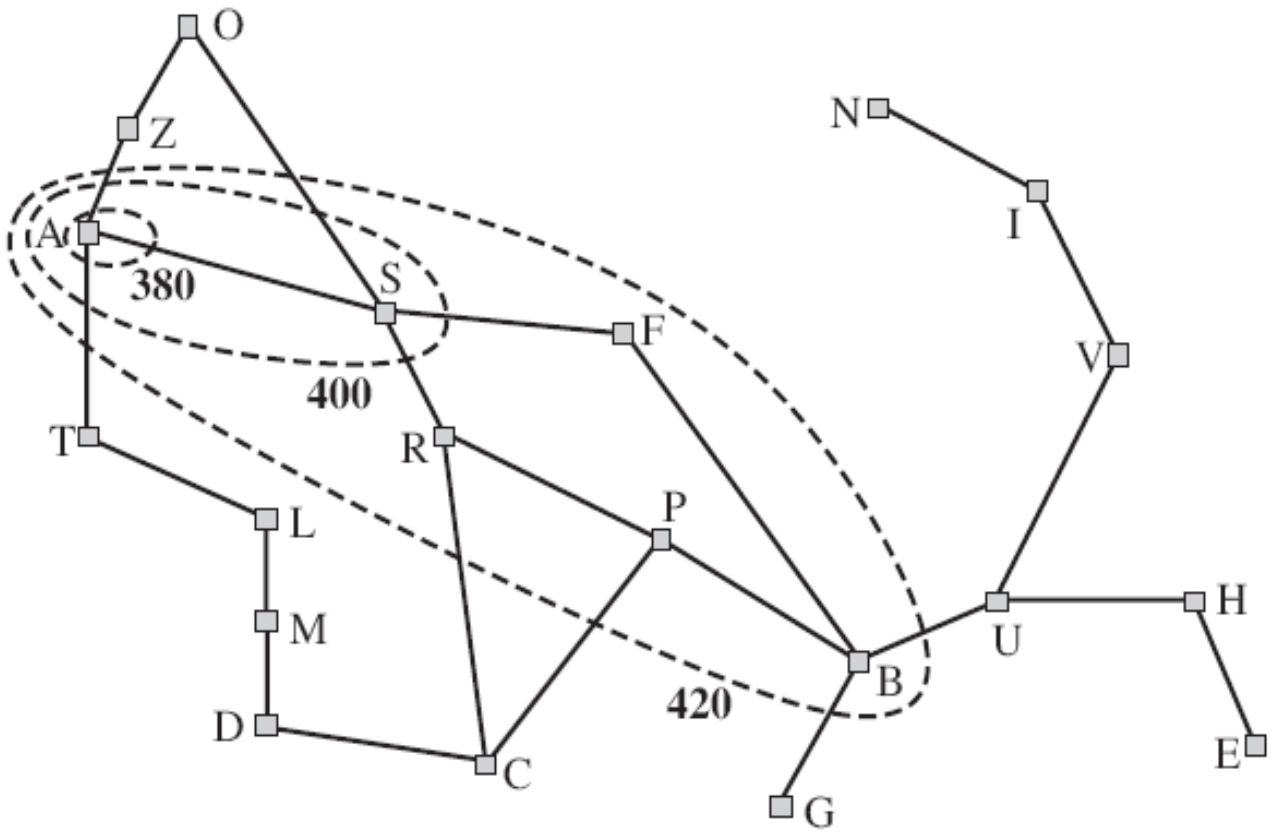


Figure 3.25 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.

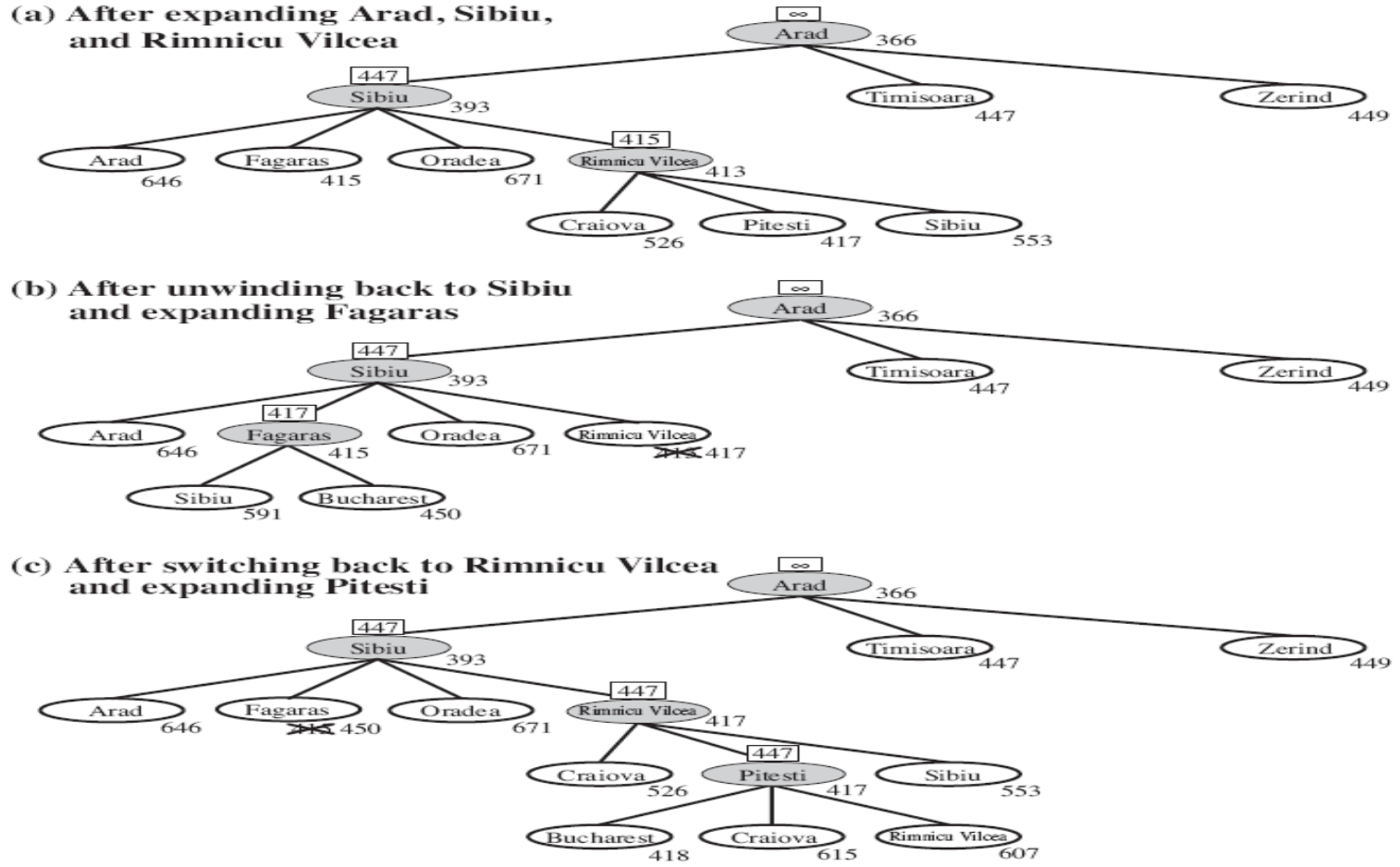


Figure 3.27 Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras). (b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best leaf value of 450. (c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path (through Timisoara) costs at least 447, the expansion continues to Bucharest.

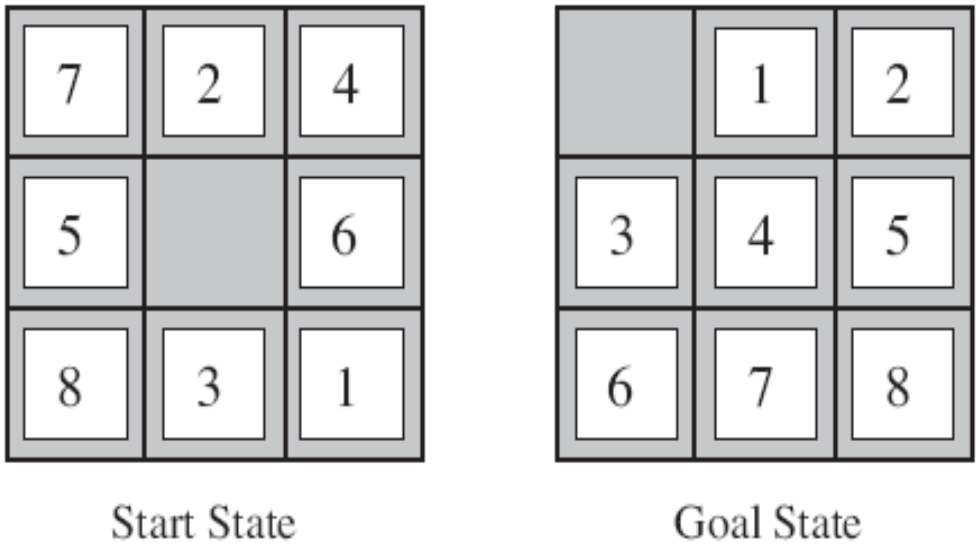


Figure 3.28 A typical instance of the 8-puzzle. The solution is 26 steps long.

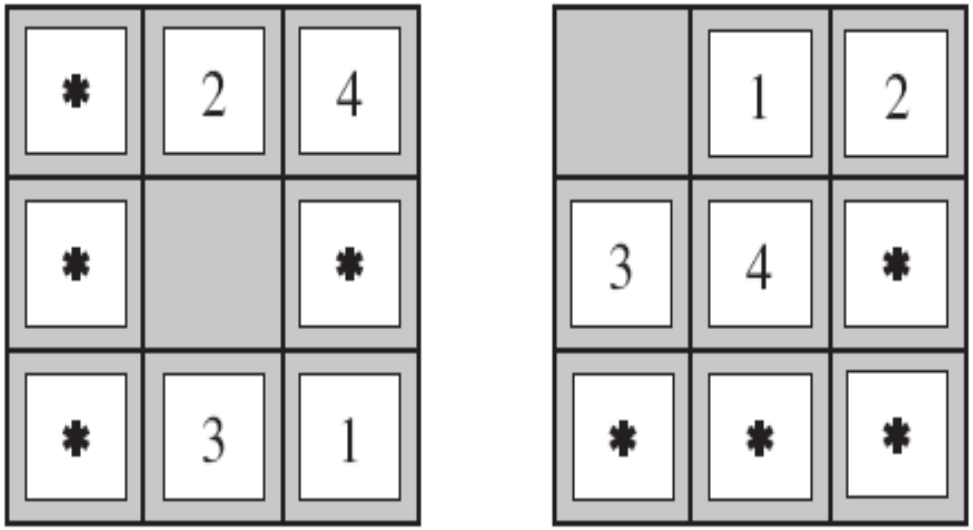


Figure 3.30 A subproblem of the 8-puzzle instance given in Figure 3.28. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

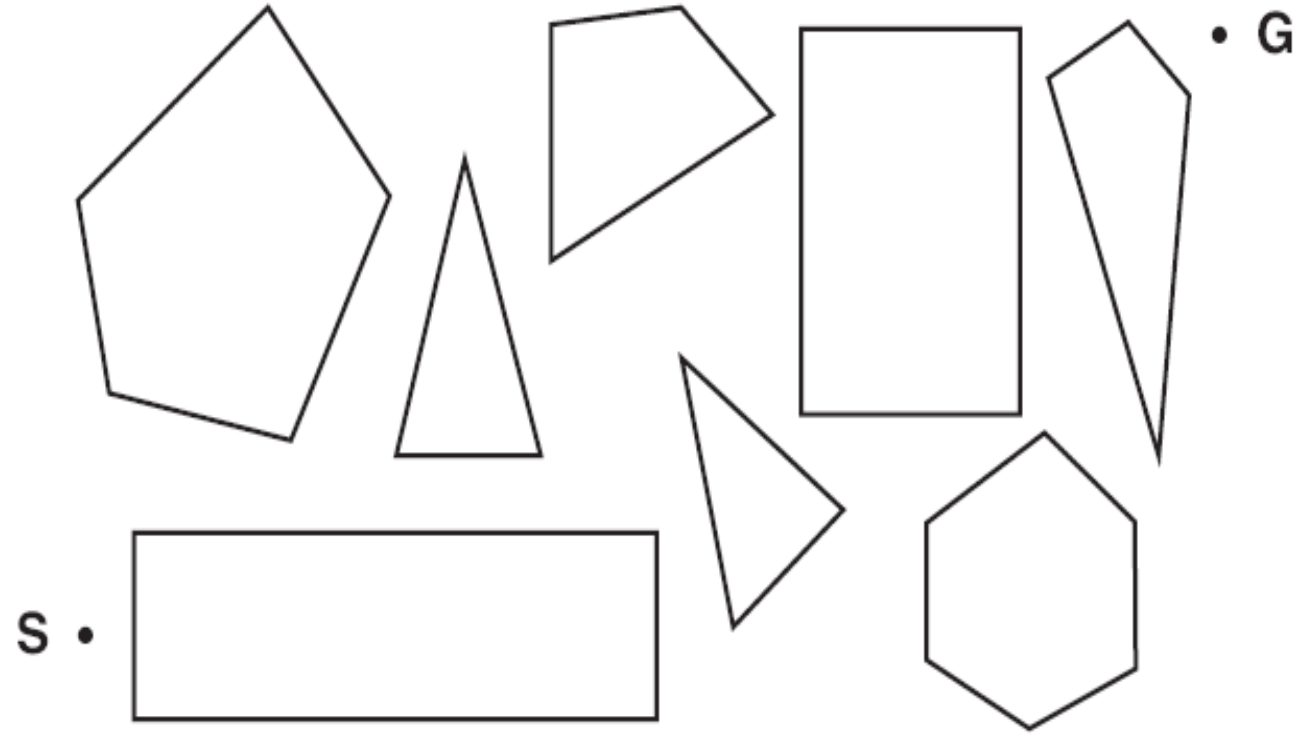


Figure 3.31 A scene with polygonal obstacles. S and G are the start and goal states.

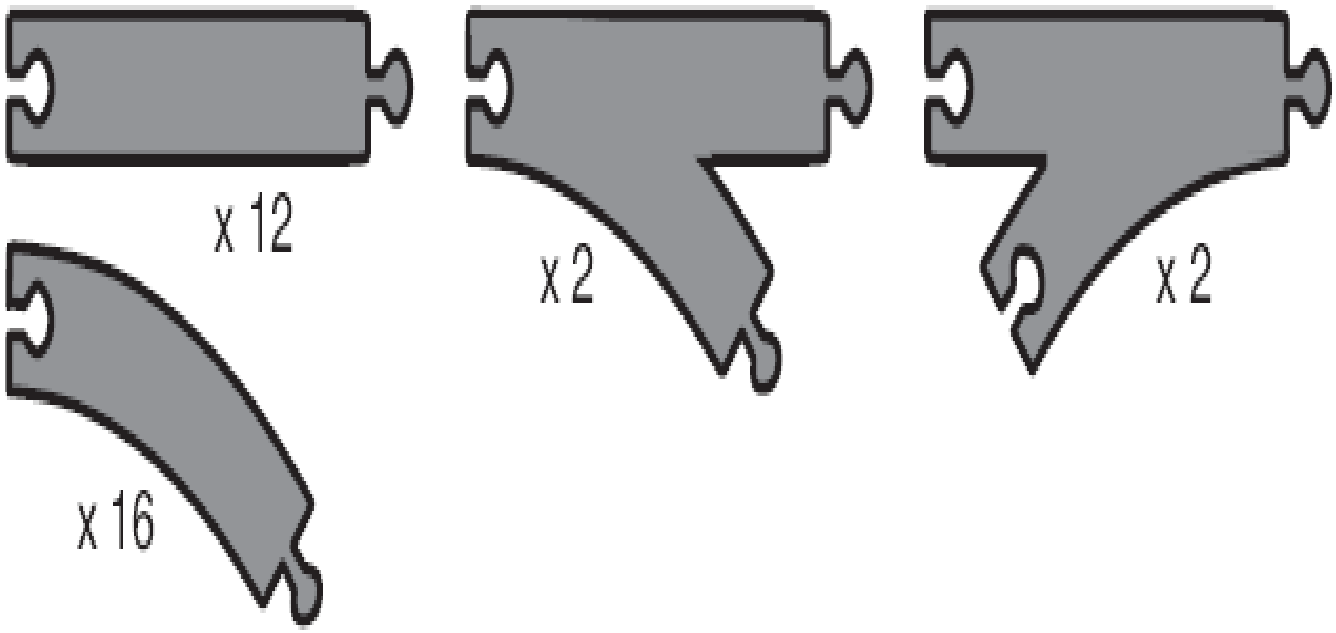


Figure 3.32 The track pieces in a wooden railway set; each is labeled with the number of copies in the set. Note that curved pieces and "fork" pieces ("switches" or "points") can be flipped over so they can curve in either direction. Each curve subtends 45 degrees.

A* Search (Informed Search) https://youtu.be/PzEWHH2v3TE

- An informed search algorithm, or a best-first search
- formulated in terms of weighted graphs
- starting from a specific start node of a graph,
- aims to find a path to the given goal node
- based on the cost of the path, cost function *g(n)*, and an estimate of the cost required to extend the path all the way to the goal, heuristic function *h(n)*.
- having the smallest cost (least distance travelled, shortest time, etc.).
- maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

A* selects the path that minimizes
$f(n) = g(n) + h(n)$

where n is the next node on the path,
$f(n)$ is the **fitness function**
$g(n)$ is the cost of the path from **the start node to** n
$h(n)$ is a **heuristic** function that estimates the cost of the **cheapest path from** n **to the goal**.

g(n) : number of moves

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

f(n):Fitness Function

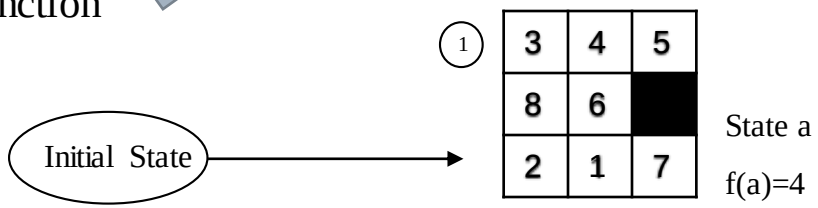
g(n):Cost Function

h(n):Heuristic Function

How many numbers are not in its goal state position?

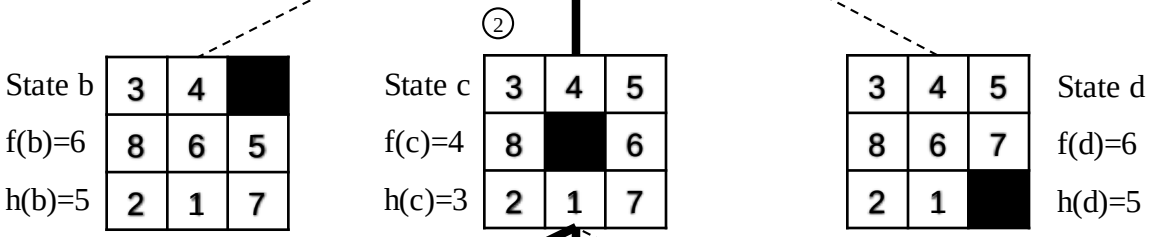
g(n)=0

level=0



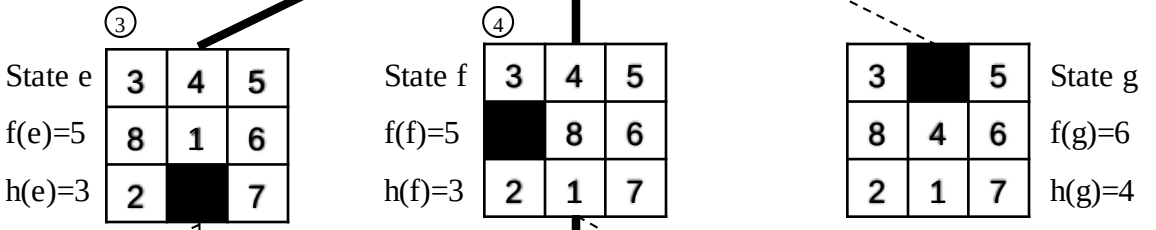
g(n)=1

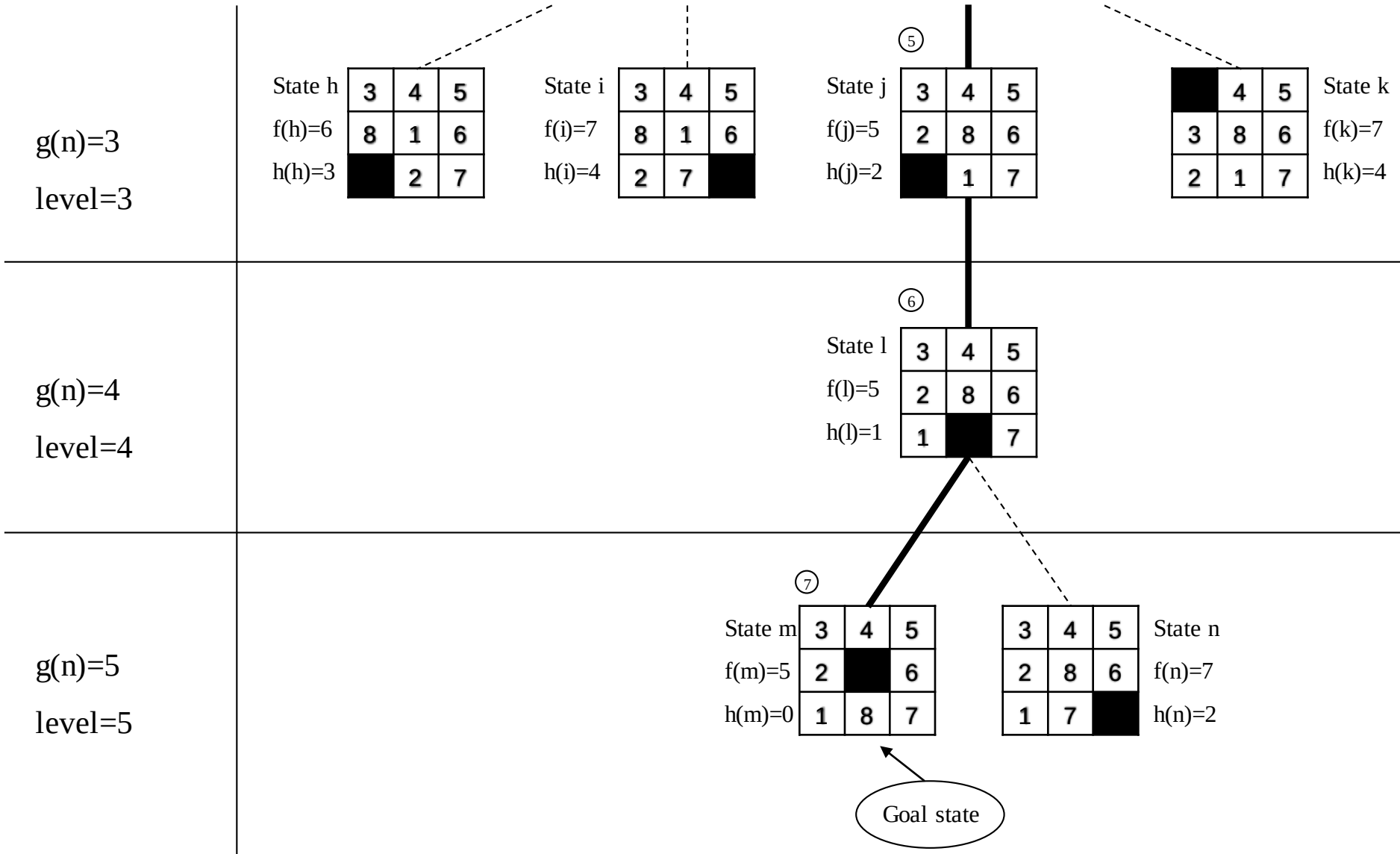
level=1

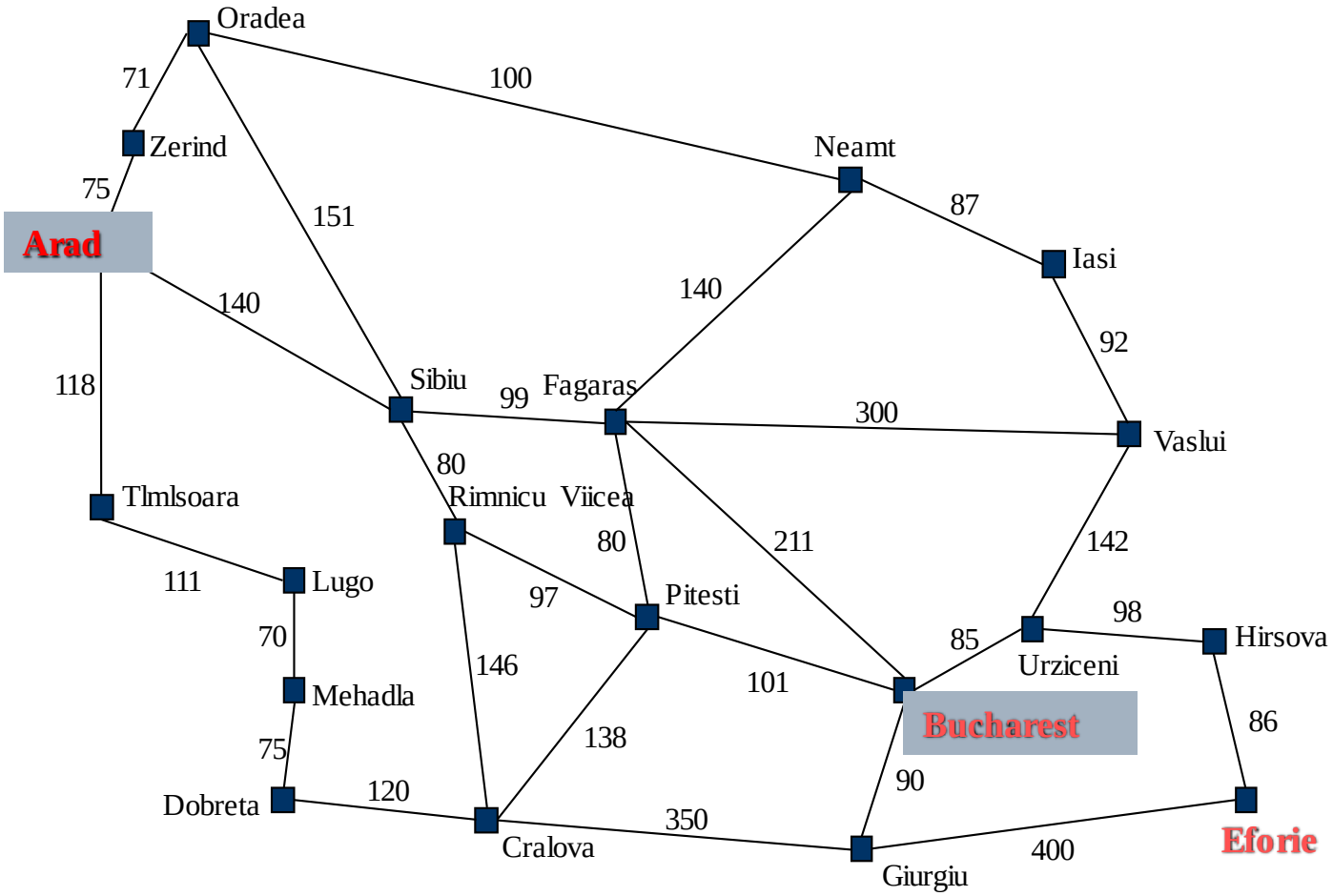


g(n)=2

level=2




g(n)=3
level=3
State h
f(h)=6
h(h)=3
State i
f(i)=7
h(i)=4
5
State j
f(j)=5
h(j)=2
State k
f(k)=7
h(k)=4
g(n)=4
level=4
6
State l
f(l)=5
h(l)=1
g(n)=5
level=5
7
State m
f(m)=5
h(m)=0
State n
f(n)=7
h(n)=2
Goal state



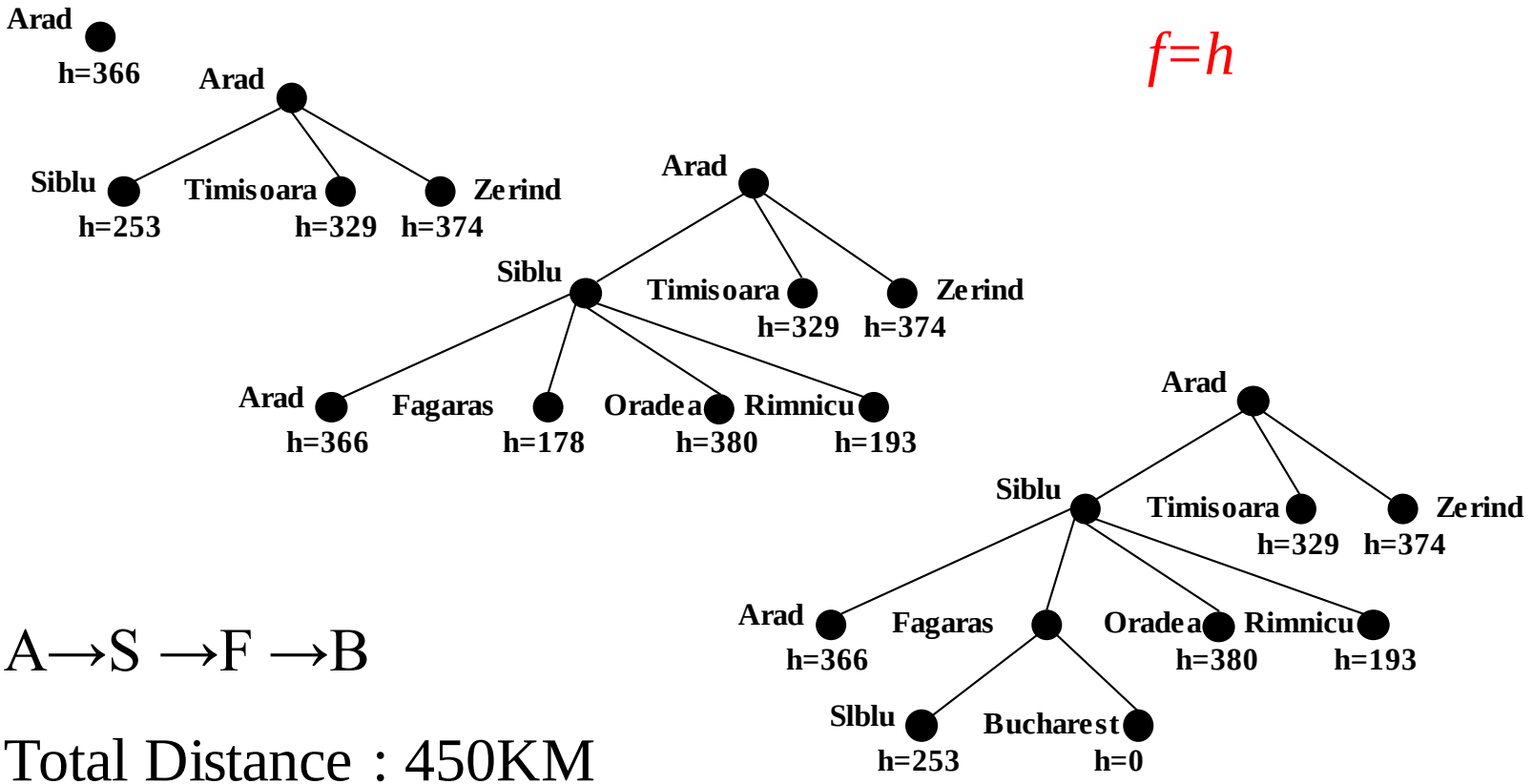
Map of Romania with road distances in km, and straight-line distances to Bucharest or Eforie.

Straight-line distance

	to B	to E
Arad	366	666
Bucharest	0	300
Craiova	160	460
Dobreta	242	542
Eforie	161	0
Fagaras	178	478
Giurgiu	77	300
Hirsova	151	86
Iasi	226	400
Lugoj	244	544
Mehadia	241	541
Neamt	234	480
Oradea	380	680
Pitesti	98	398
Rimnicu Vilcea	193	493
Sibiu	253	553
Timisoara	329	629
Urziceni	80	220
Vaslui	199	230
Zerind	374	674

Greedy Search

f=h



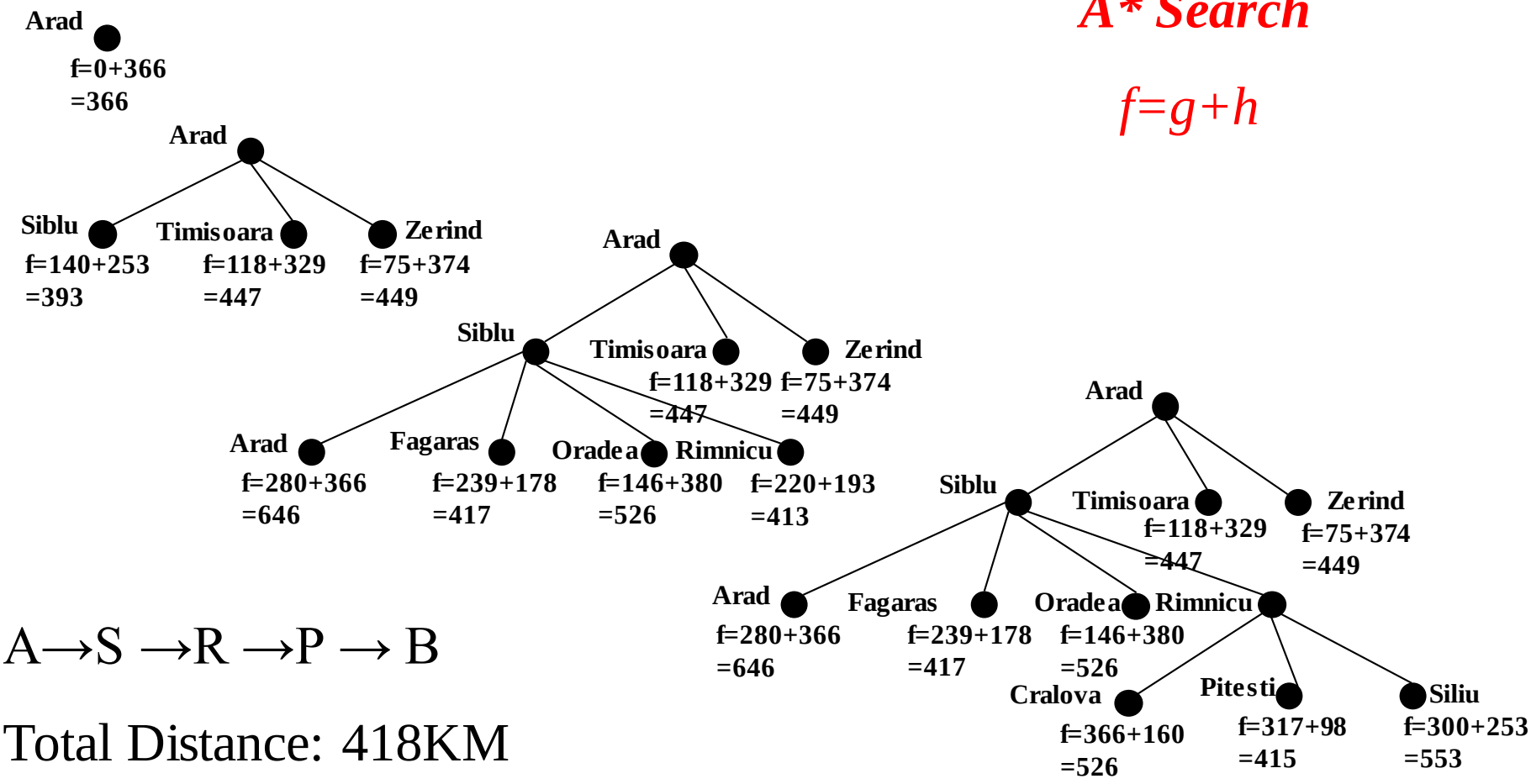
A→S →F →B

Total Distance : 450KM

Stages in greedy search for Bucharest, using the straight-line distance to Bucharest as the heuristic function h_{SLD}. Nodes are labelled with their *h*-values.

A* Search

$f=g+h$



A→S →R →P → B

Total Distance: 418KM

Stages in an A* search for Bucharest. Nodes are labelled with $f=g+h$. The h values are the straight-line distances to Bucharest.

g(n):path cost from start node to node n.

Homework

Use the map of Romania with "road distances" and "straight-line distances to E(Eforie)" to find the shortest path from city Arad to Eforie by

(a) Greedy search

(b) A* search

For each search method, draw the search tree.

Homework

Solve the 8-puzzle problem with the initial and goal state given below, draw the search tree.
(Define your heuristic function such as "How many numbers are not in its goal state position?" or "How many steps needed for the numbers to move to their correct position" …etc.)

Initial state

3	5	7
6		4
2	1	8

Goal state

3	4	5
2		6
1	8	7

Game

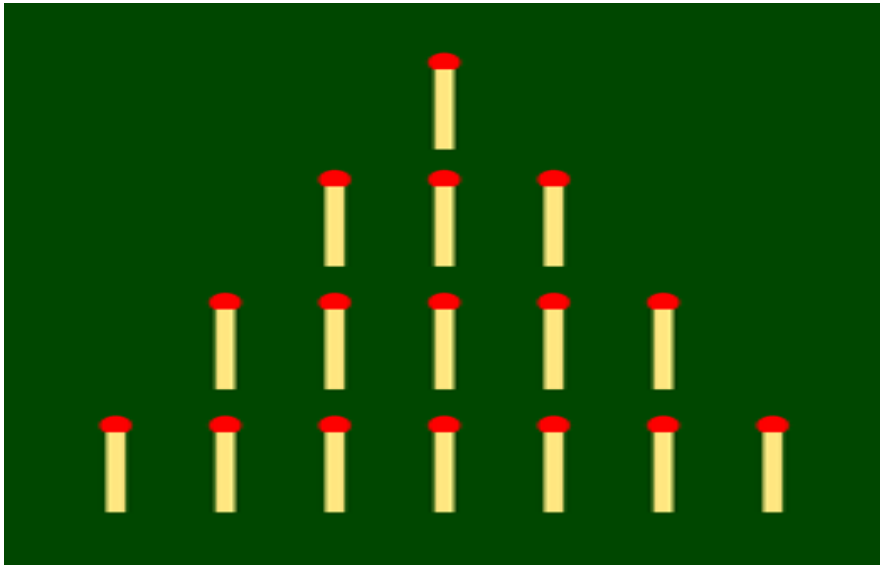
- A **game** is a structured form of play, usually undertaken for entertainment or fun, and sometimes used as an educational tool.
- A **game tree** is a directed graph whose nodes are positions in a game and whose edges are moves.
- The **complete game tree** for a game is the game tree starting at the initial position and containing all possible moves from each position; the complete tree is the same tree as that obtained from the extensive-form game representation.

https://en.wikipedia.org/wiki/Game

Nim

- Two players take turns removing (or "nimming") objects from distinct heaps or piles.
- On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap or pile.
- The goal of the game is either to avoid taking the last object or to take the last object.

Class practice:
Draw the complete game tree of 5 matches Nim



Matches set up in rows for a game of Nim. Players take turns to choose a row and remove any number of matches from it.

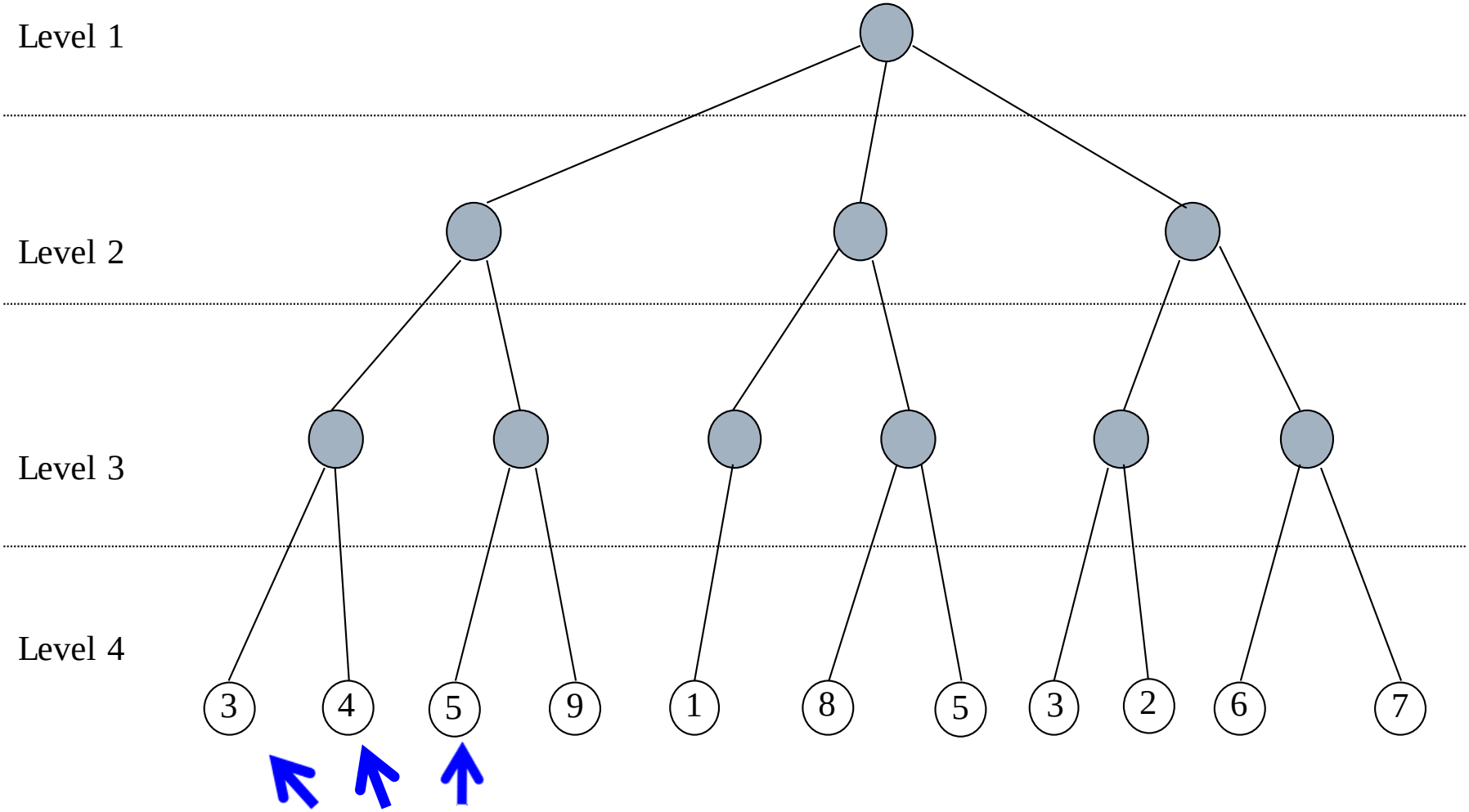
Class practice: two classmates play 7 matches Nim

Minimax Tree

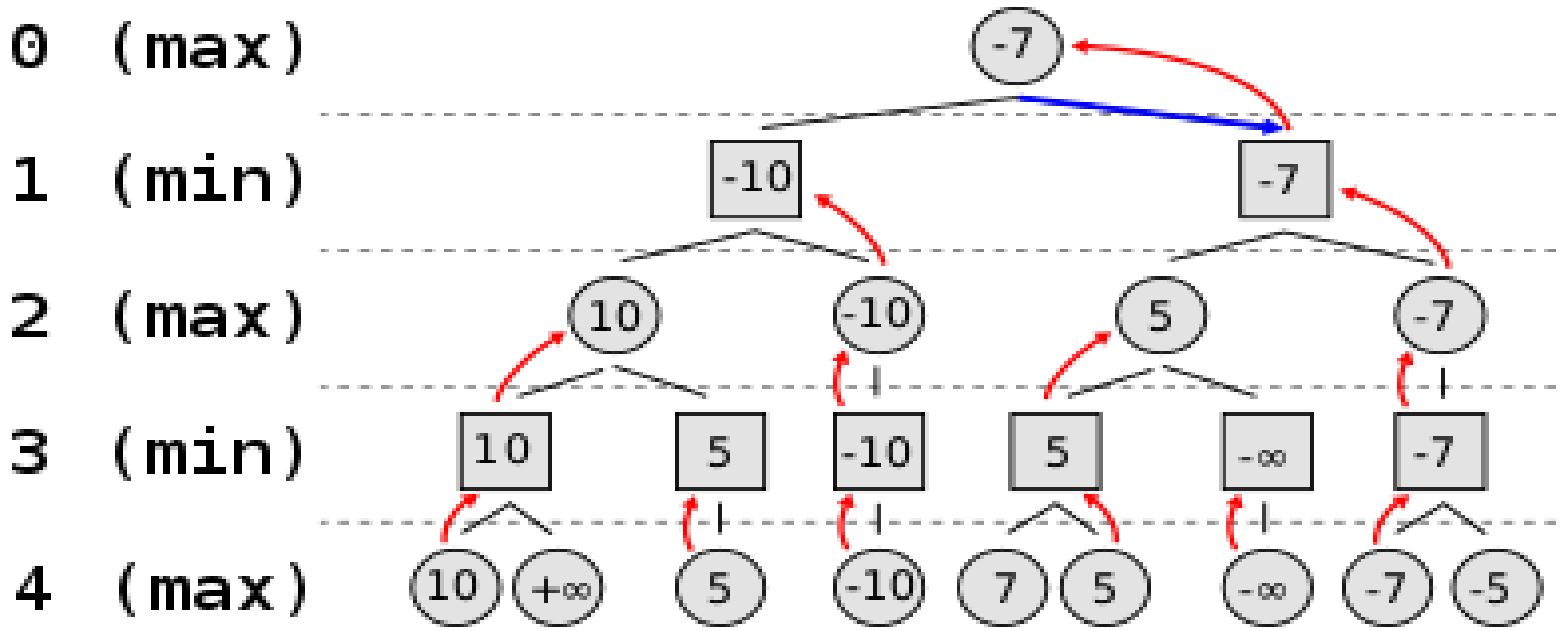
- minimizing the possible loss for a worst case (maximum loss) scenario
- n-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves

- The minimax function returns a heuristic value for leaf nodes (terminal nodes and nodes at the maximum search depth).
- The heuristic value is a score measuring the favorability of the node for the maximizing player.
- Favorable outcome, such as a win, for the maximizing player have higher scores than nodes more favorable for the minimizing player.2
- The heuristic value for terminal (game ending) leaf nodes are scores corresponding to win, loss, or draw, for the maximizing player. 3

Game tree with maximum search depth of 3 levels



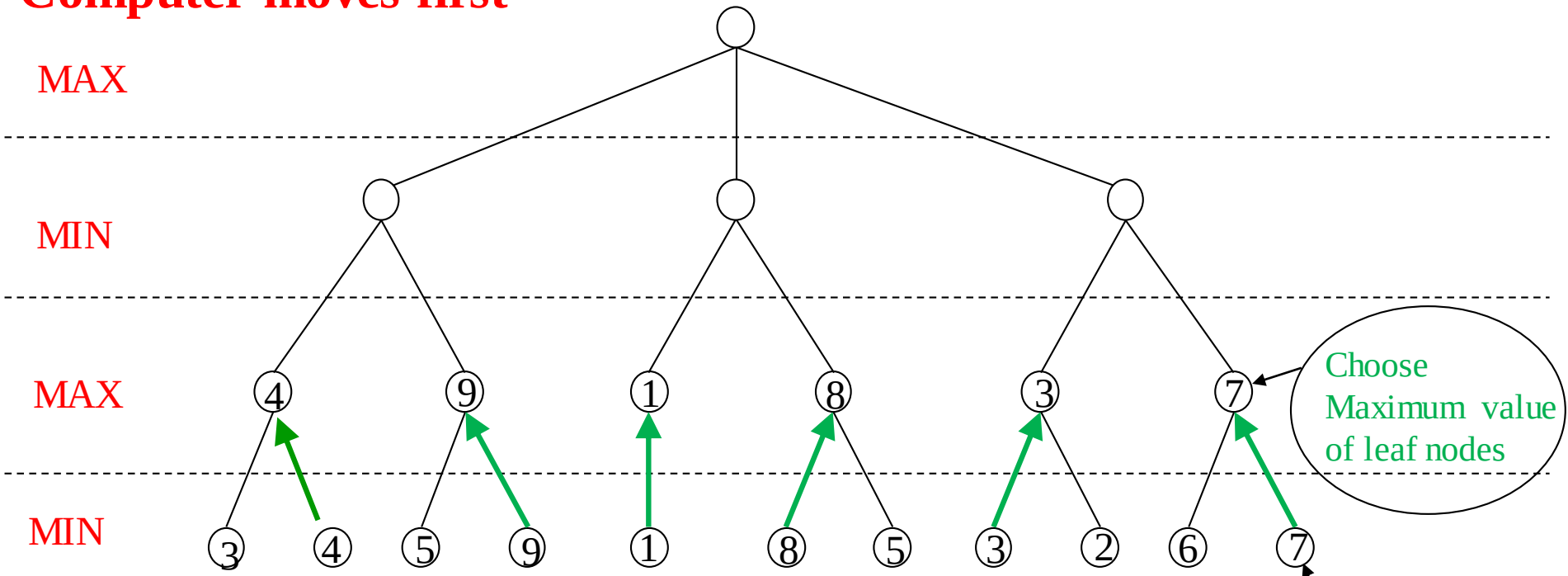
Heuristic function value for terminal (game ending) leaf nodes



- The circles represent the moves of the player running the algorithm (*maximizing player*), and squares represent the moves of the opponent (*minimizing player*)
- The moves where the maximizing player wins are assigned with positive infinity, while the moves that lead to a win of the minimizing player are assigned with negative infinity.

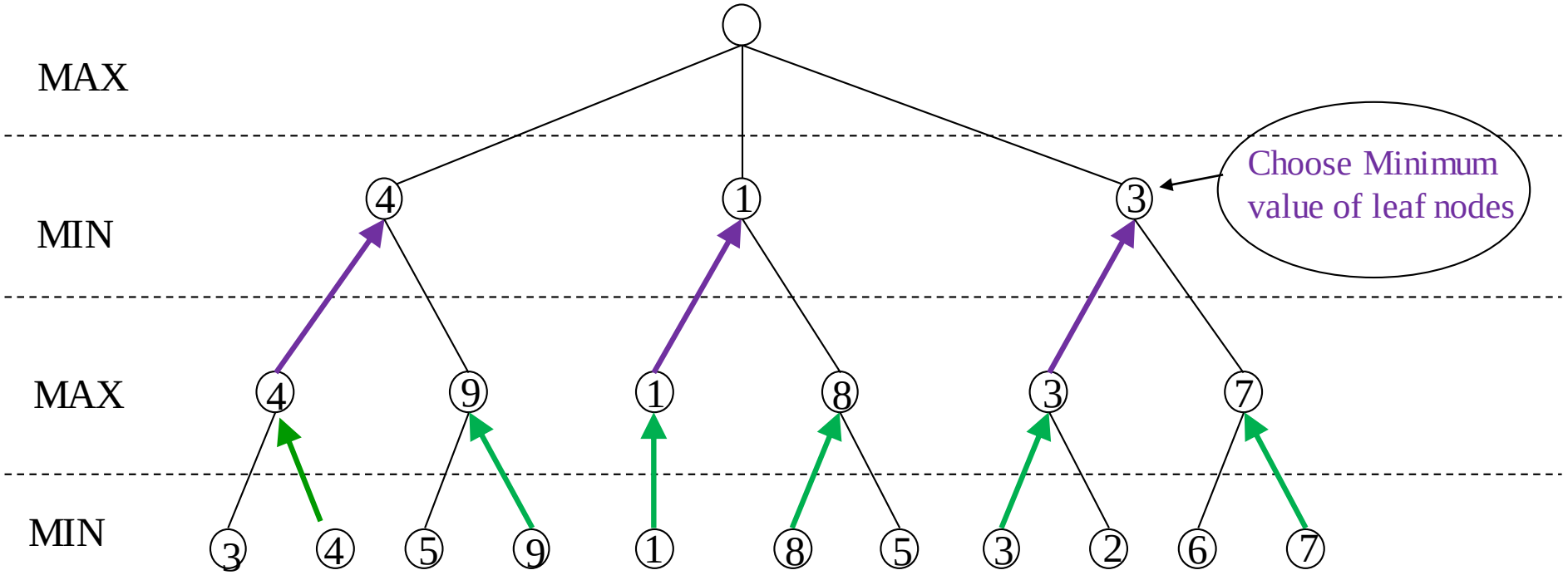
https://en.wikipedia.org/wiki/Minimax#/media/File:Minimax.svg

Computer moves first



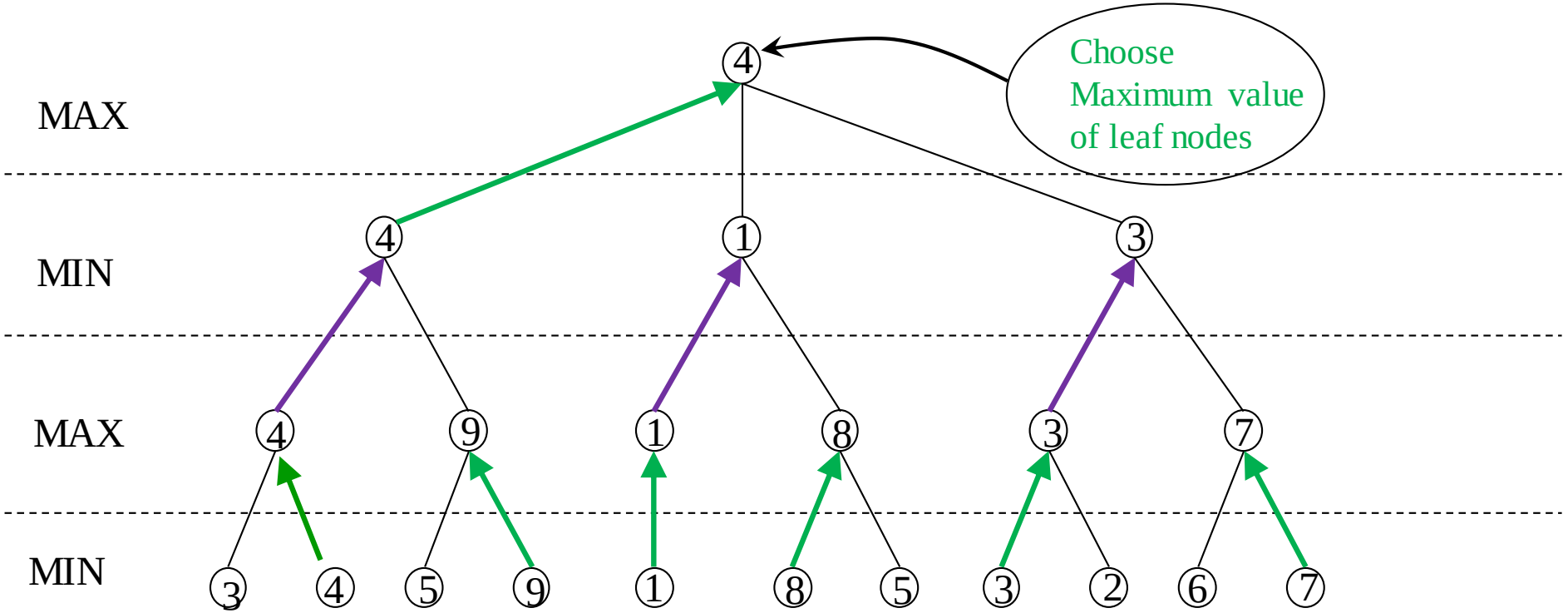
MiniMax Process :

Max (Computer) vs Min (Player)



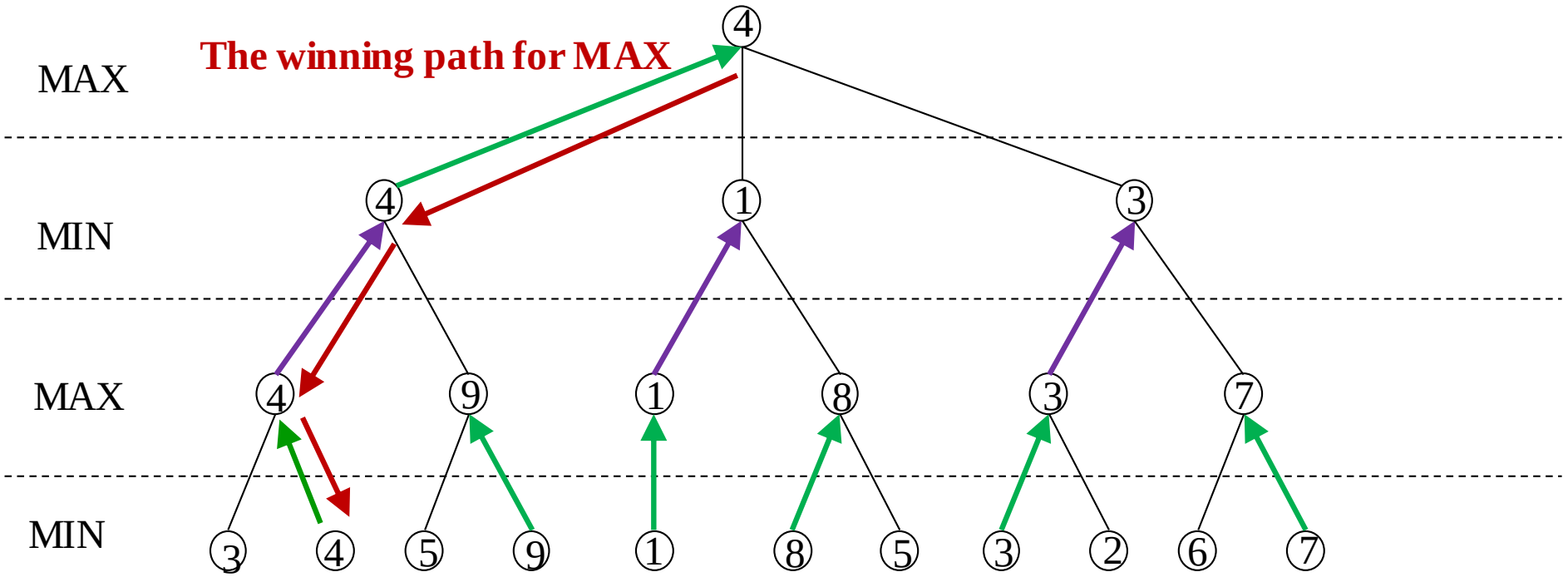
MiniMax Process :

Max (Computer) vs Min (Player)



MiniMax Process :

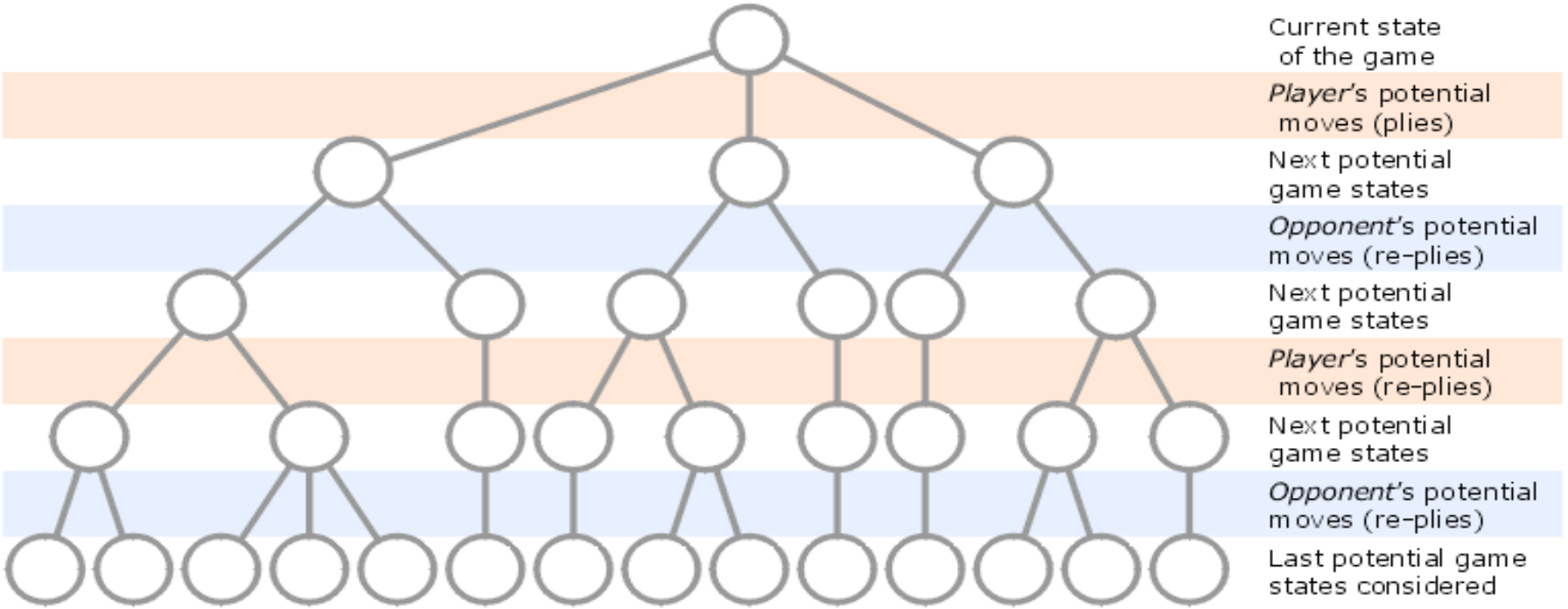
Max (Computer) vs Min (Player)



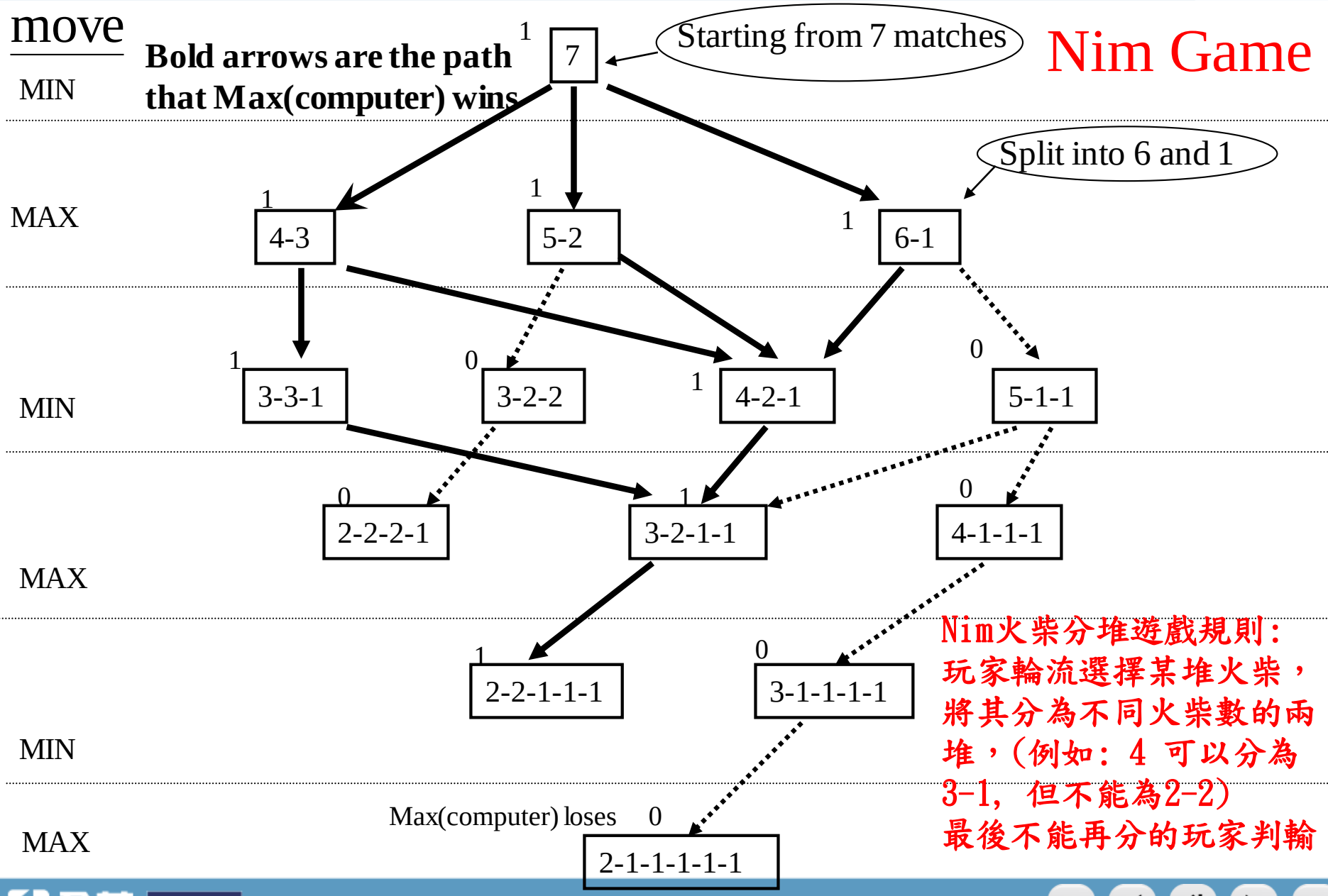
MiniMax Process :

Max (Computer) vs Min (Player)

Minimax on a two-person game tree of 4 plies (step by step)

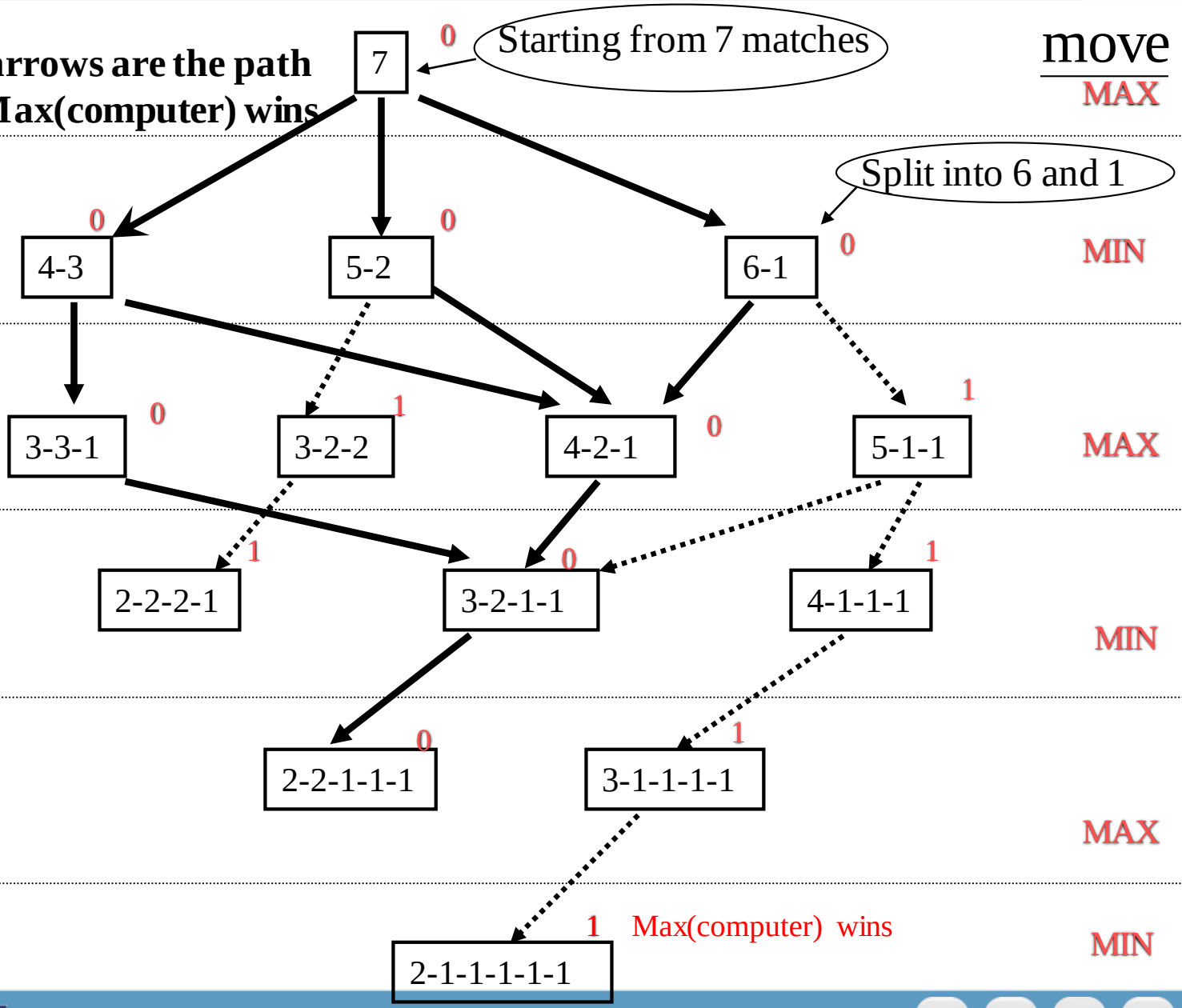


Nim Game



Nim火柴分堆遊戲規則：
玩家輪流選擇某堆火柴，將其分為不同火柴數的兩堆，(例如：4 可以分為3-1，但不能為2-2)
最後不能再分的玩家判輸

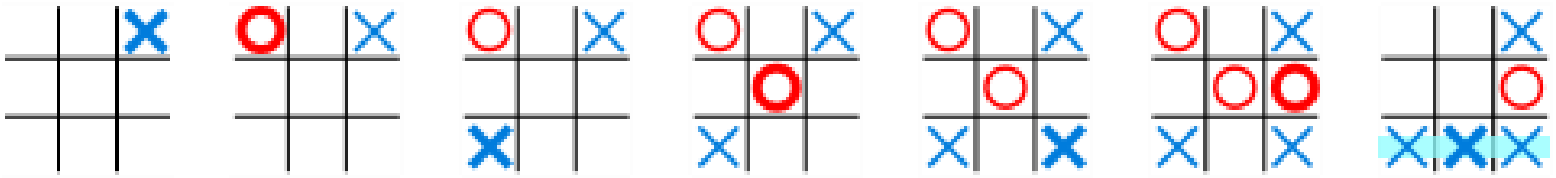
Bold arrows are the path that Max(computer) wins

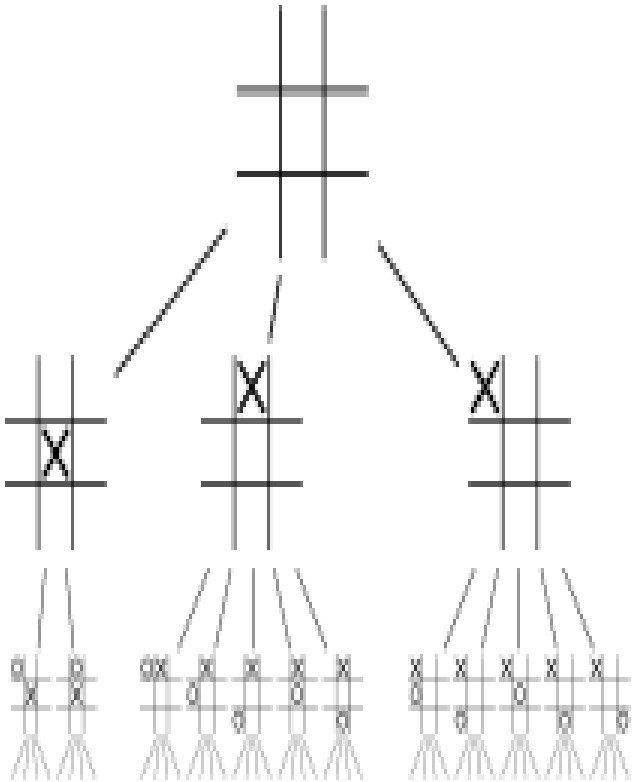


Tic-tac-toe game

Tic-tac-toe noughts and crosses , or **Xs and Os** a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid.

Player who places three of their marks in a diagonal, horizontal, or vertical row is the winner.





- The diagram shows first two levels, or plies, in the game tree for tic-tac-toe.
- **Rotations and reflections of positions** are equivalent, so the first player has three choices of move.
- The second player has two choices for the reply if the first player played in the center, otherwise five choices. And so on.
- The number of leaf nodes in the complete game tree is the number of possible different ways the game can be played.
- The game tree for tic-tac-toe has totally **255,168** leaf nodes.

- One way to pick the best move in a game is to search the game tree using any of numerous tree search algorithms, combined with minimax-like rules to prune the tree.
- The game tree for tic-tac-toe is easily searchable, but the complete game trees for larger games like chess are much too large to search.
- Instead, a chess-playing program searches a partial game tree: typically as many plies from the current position as it can search in the time available.
- Increasing the search depth (i.e., the number of plies searched) generally improves the chance of picking the best move.

Two-person games can also be represented as and-or trees.

For the first player to win a game, there must exist a winning move for all moves of the second player.

This is represented in the and-or tree by using disjunction to represent the first player's alternative moves and using conjunction to represent all of the second player's moves.

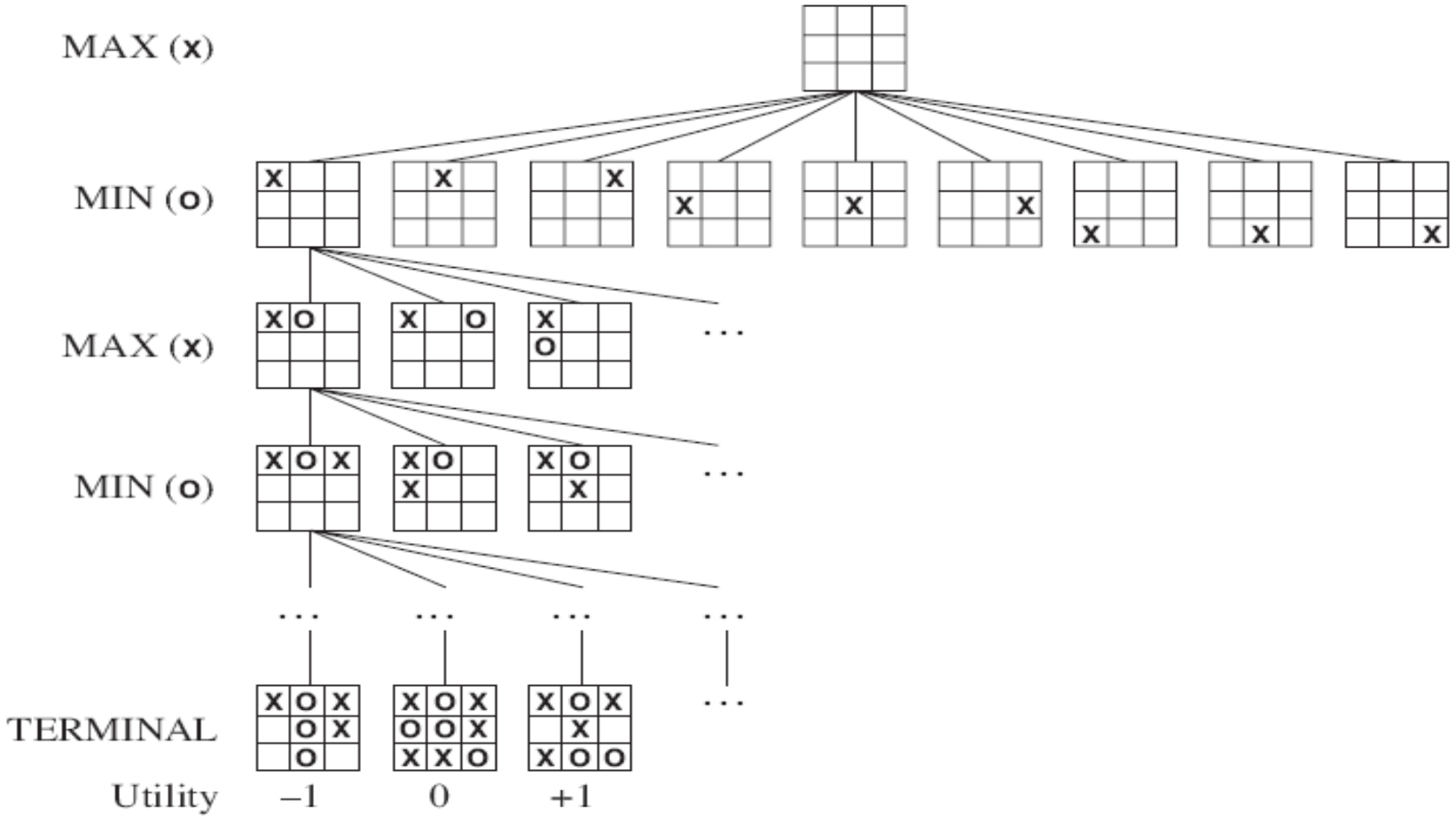
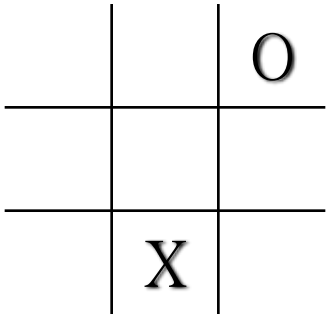
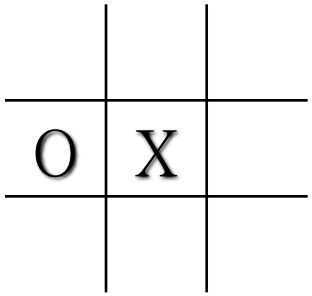
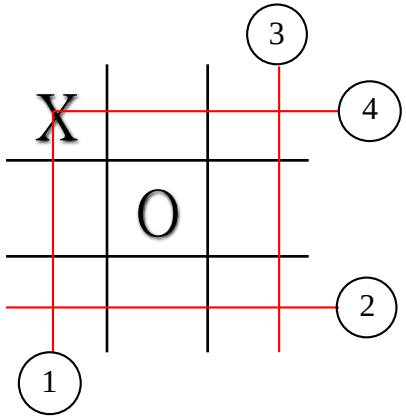


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

A heuristic for Tic-Tac-Toe: count the number of possible ways to connect 3 marks in a row, column or diagonal line.







O: 6

X: 5

f(n)=6-5=1

O: 4

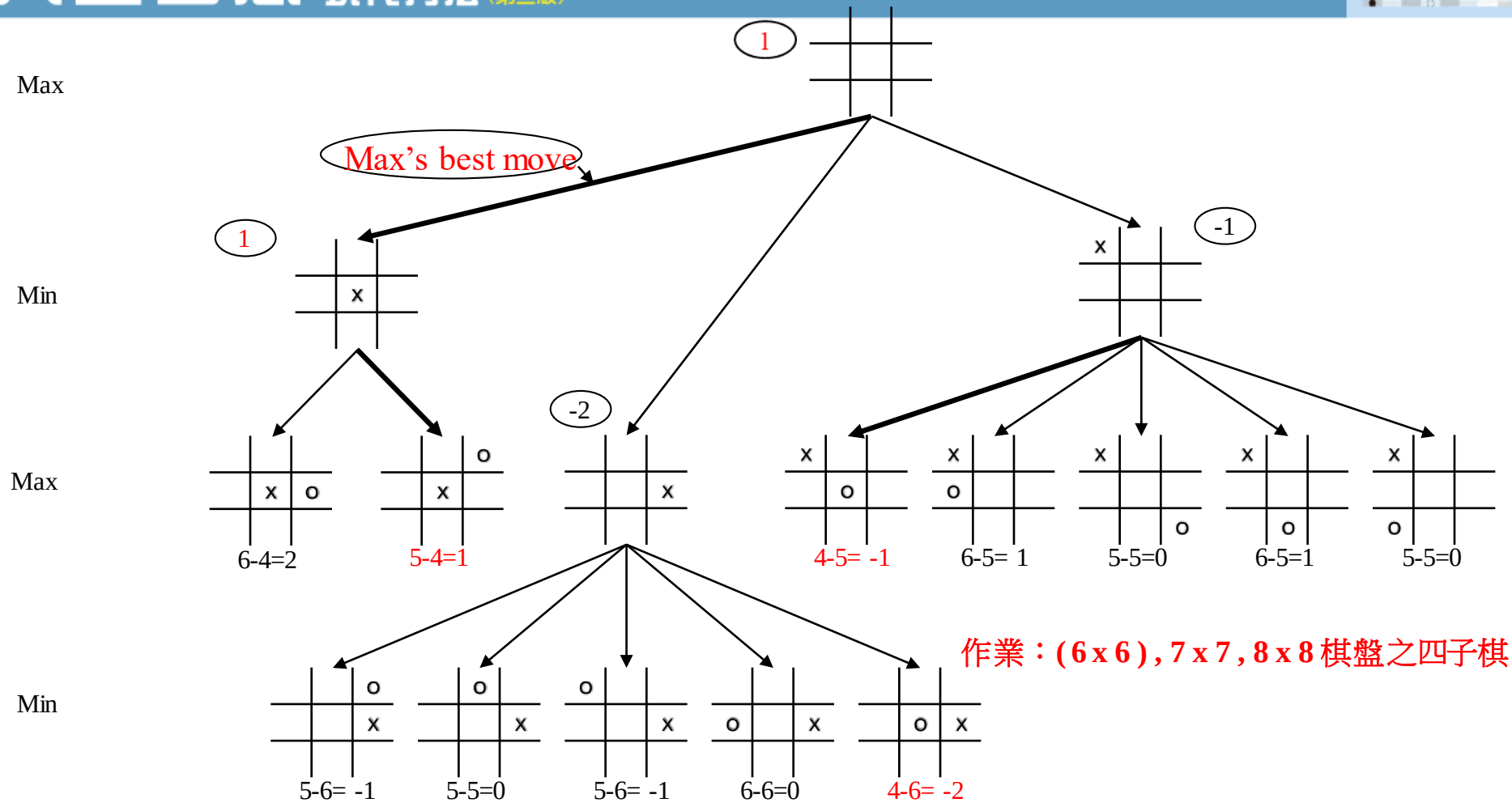
X: 6

f(n)=4-6=-2

O: 5

X: 4

f(n)=5-4=1



作業：(6x6),7x7,8x8棋盤之四子棋

Tic-Tac-Toe for considering the search depth of 2

a *heuristic evaluation function:*
returns a relative score, e.g., +∞ for computer-win, -∞ for opponent-win, 0 for neutral, and a number in between to indicate the relative advantage of the computer vs. the opponent.
In Tic-Tac-Toe, a possible heuristic evaluation function for the current board position is:
+100 for EACH 3-in-a-line for computer.
+10 for EACH two-in-a-line (with a empty cell) for computer.
+1 for EACH one-in-a-line (with two empty cells) for computer.
Negative scores for opponent, i.e., -100, -10, -1 for EACH opponent's 3-in-a-line, 2-in-a-line and 1-in-a-line.
0 otherwise (empty lines or lines with both computer's and opponent's seeds).
https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe_AI.html

beginning situation has values as below:

3	2	3
2	4	2
3	2	3

a later one can be like this (X is moving now)

X	10	O
O	O	110
X	20	20

Group Homework: develop your heuristic function for a tic-tac-toe connecting 4 same marks in a 6x6 grid and implement it using any computer program.

	O				
		O	X		
		X	O		
				X	

Alpha-Beta Prunning

- Some of the branches of the game tree won't be taken if playing against an intelligent opponent
- Pruning can be used to ignore some branches
- α= best already explored option along the path to the root for **MAX**, including the current node
- β= best already explored option along the path to the root for **MIN**, including the current node

Implementation of Cutoffs

At **MAX** node, v= largest value from its children visited so far; cutoff if $v \geq \beta$

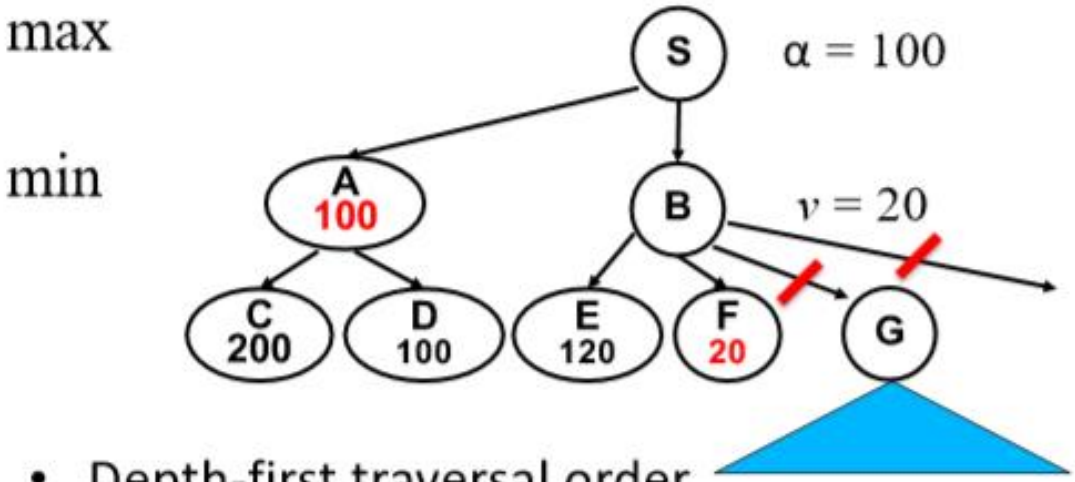
- v value at MAX comes from its descendants
- β value at MAX comes from its MIN node ancestors

At **MIN** node, v= smallest value from its children visited so far; cutoff if $v \leq \alpha$

- α value at MIN comes from its MAX node ancestors
- v value at MIN comes from its descendants

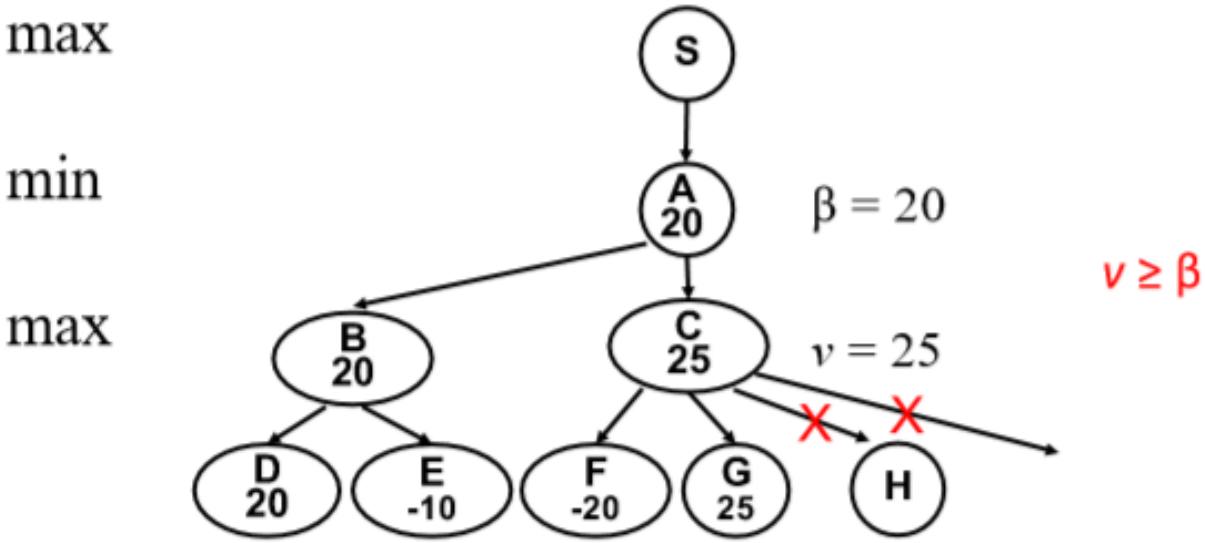
Pruning happens when there is $\alpha \geq \beta$.

Alpha-Beta Idea: Alpha Cutoff



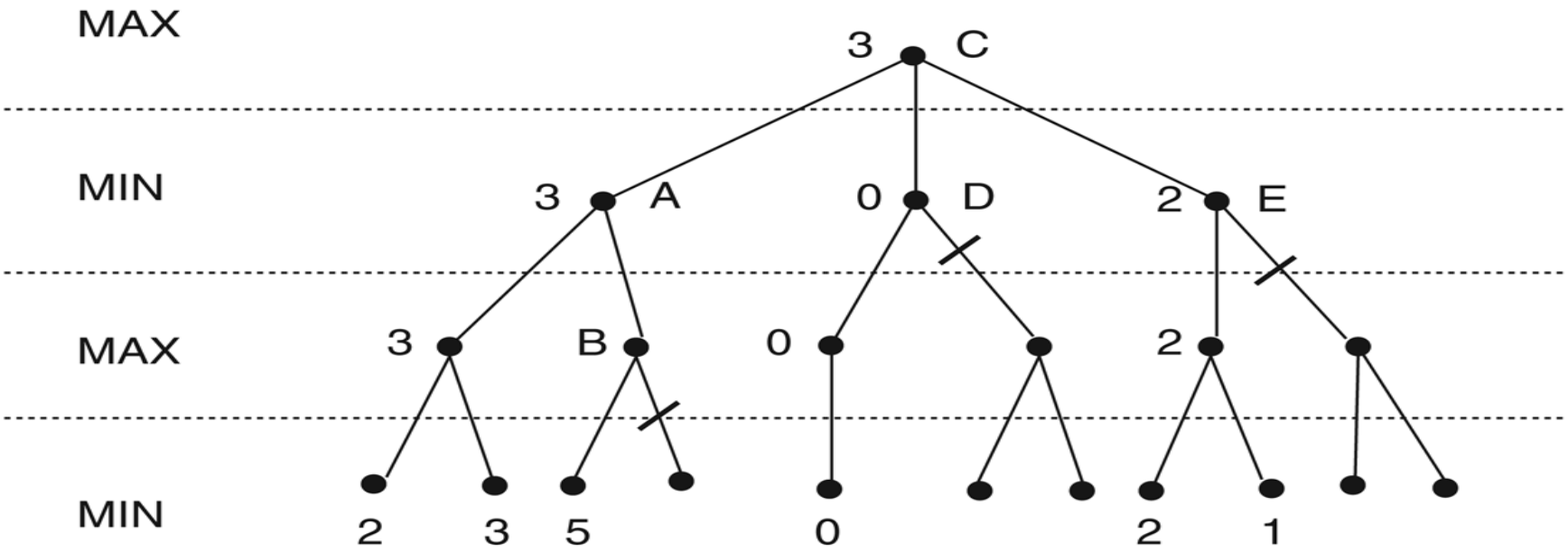
- Depth-first traversal order
- After returning from A, can get ***at least 100 at S***
- After returning from F, can get ***at most 20 at B***
- At this point no matter what minimax value is computed at G, S will prefer A over B. So, S loses interest in B
- There is no need to visit G. The subtree at G is pruned. Saves time. Called "**Alpha cutoff**" (at MIN node B)

Beta Cutoff Example



- After returning from B, can get ***at most 20 at MIN node A***
- After returning from G, can get ***at least 25 at MAX node C***
- No matter what minimax value is found at H, A will NEVER choose C over B, so don't visit node H
- Called "**Beta Cutoff**" (at MAX node C)

Alpha-Beta Pruning
(States without numbers are not evaluated)



A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

Homework:

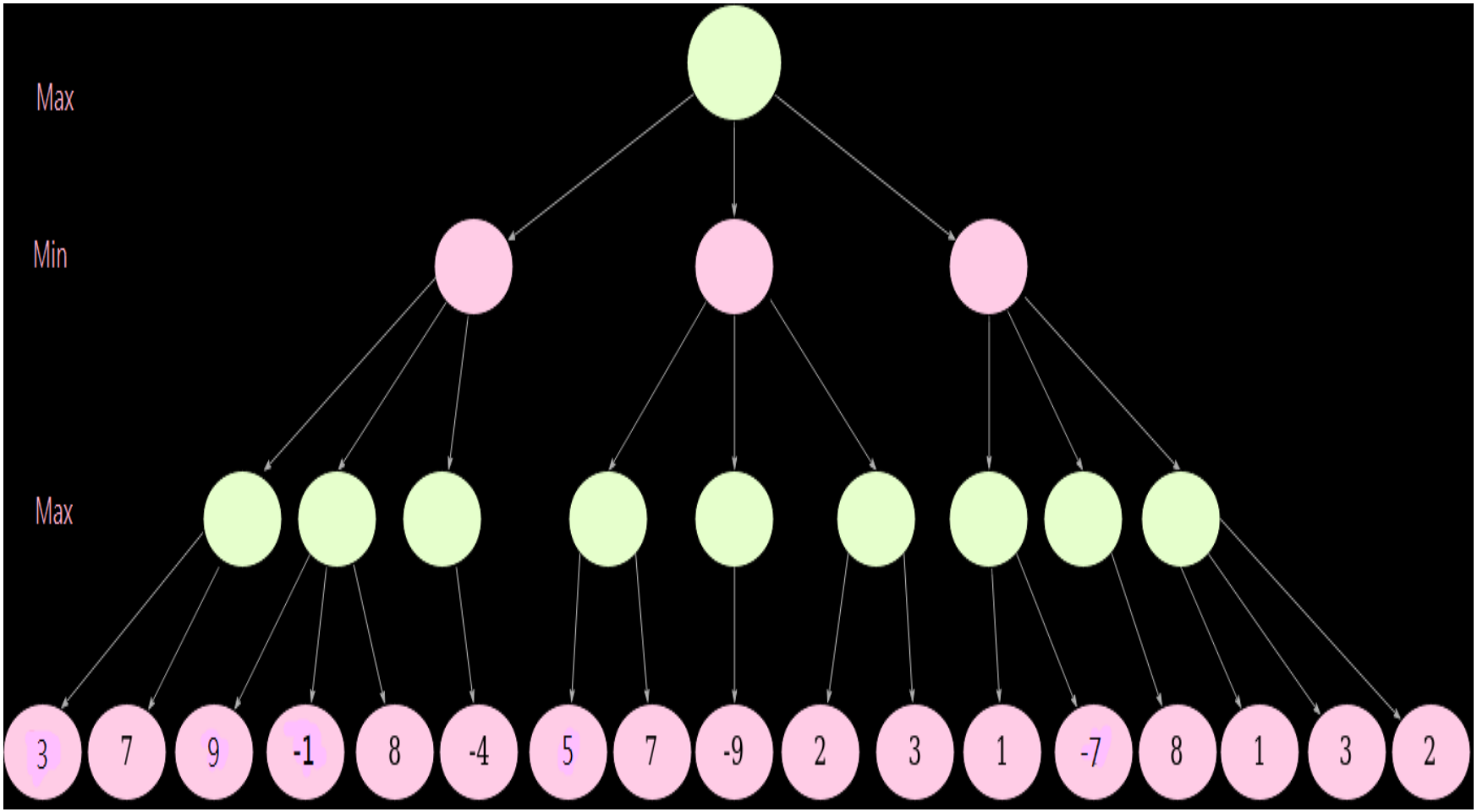
Assume that the following tree is the state space of a game with 3 more moves.

Identify the branches that can be pruned in the game tree using alpha-beta pruning.

Try to solve the problem using any computer language or software.

You may refer to:

https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/


Max
Min
Max
3
7
9
-1
8
-4
5
7
-9
2
3
1
-7
8
1
3
2