MODULE 3

PROBLEM SOLVING

Refer Chapter 3 & 4

3	**Problem solving**		**15**	**CO2**
	3.1	Solving problem by Searching : Problem Solving Agent, Formulating Problems, Example Problems.		
		*Defining problem as state space search, production rules, Problem characteristics, issues in design of search program,		
	3.2	Uninformed Search Methods: Breadth First Search, Depth First Search, Depth Limited Search, Iterative Deepening depth first search		
	3.3	Informed Search Methods: Heuristic, properties of good heuristic, Greedy best first Search, A* Search, AO* search.		
	3.4	**Local Search Algorithms and Optimization Problems:** Hill-climbing search- Hill climbing algorithm, problems and solutions in hill climbing Constraint satisfaction- Defining CSP, inferences in CSP, CSP Backtracking algorithm* Genetic algorithms*: The genetic algorithm process, solving problems with GA for optimization and learning, significance of genetic operators		
		Adversarial Search: Games, Optimal strategies, The minimax algorithm , Alpha-Beta Pruning,		
		#Self Learning – Online search algorithms, partially observable/imperfect information games		

Solving problem by **SEARCHING**

- An agent can find a **sequence of actions** that achieves its goals when no single action will do.
- **Problem solving begin**s with precise definitions of PROBLEMS AND THEIR SOLUTIONS
  - **Uninformed search algorithms**—algorithms that are given **no information** about the problem other than its definition
  - **Informed search algorithms**—Can work well given some **guidance** on where to look for solutions.

Search Algorithm Terminologies

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space.
- A search problem can have three main factors:
  - **Search Space:** Search space represents a set of **possible solutions**, which a system may have.
  - **Start State:** It is a state from where agent **begins the search**.
  - **Goal test:** It is a function which observe the current state and returns whether the **goal state is achieved or not.**
- **Search tree:** A tree representation of search problem is called Search tree.
- **Actions:** It gives the **description of all the available actions** to the agent.
- **Transition model:** Representation of description of what each action does.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

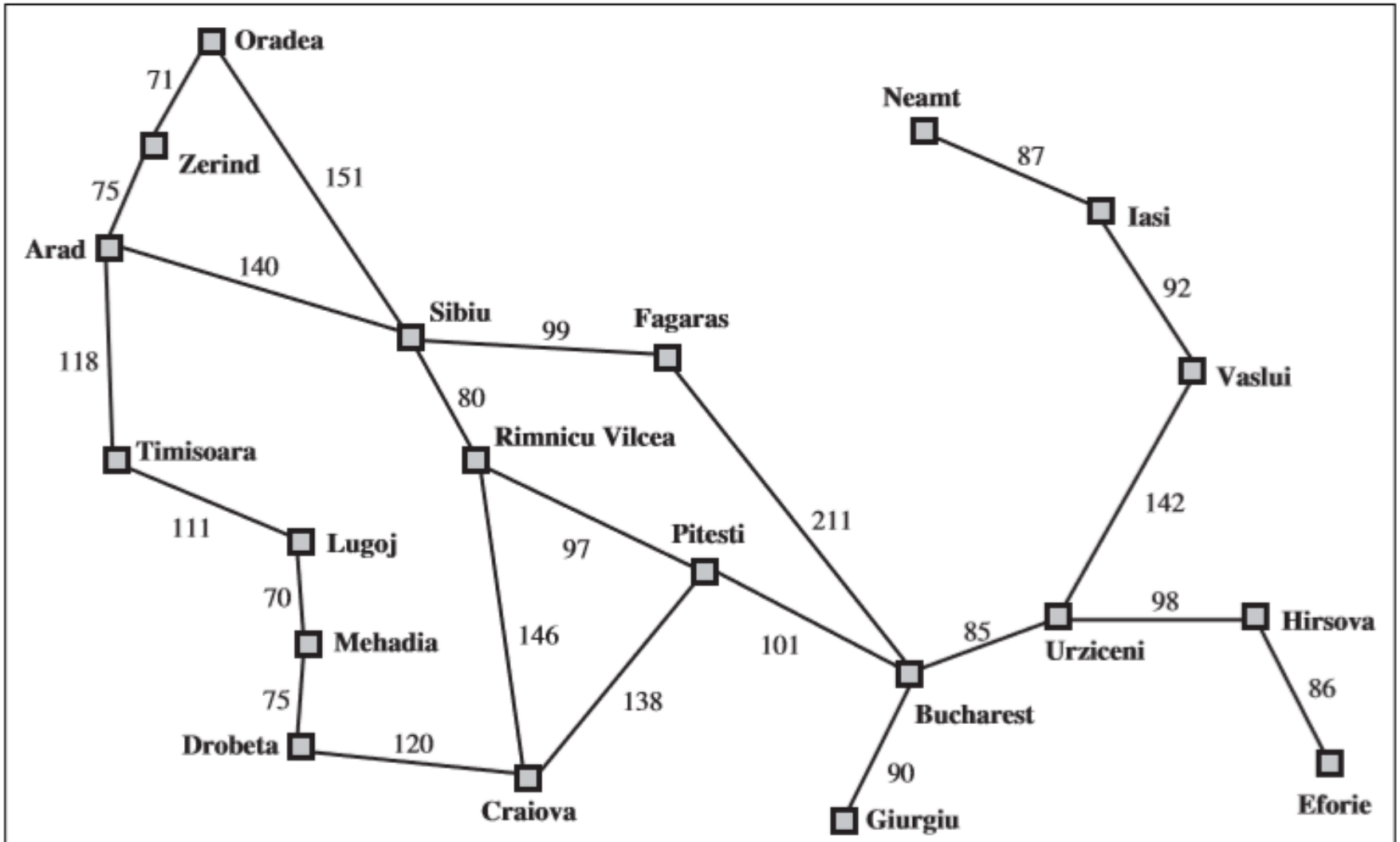


Figure 3.2 A simplified road map of part of Romania.

PROBLEM-SOLVING AGENTS

- **GOAL FORMULATION**:- Based on the **current situation** and the agent's performance measure.
- **PROBLEM FORMULATION**:- Process of **deciding what actions** and states to **consider**, given a **GOAL**
- "**FORMULATE, SEARCH, EXECUTE**" design for the agent

EXAMPLE PROBLEMS

(Toy and real-world problems)

A TOY PROBLEM is intended to illustrate or exercise various problem-solving methods given a **concise, exact description** and hence is usable by different researchers to compare the performance of algorithms.

REAL-WORLD PROBLEM is one whose solutions people actually care about-Eg healthcare

Example Problems

- Toy problems
  - Illustrate/test various problem-solving methods
  - Concise, exact description
  - Can be used to compare performance

  EXAMPLES: 8-puzzle, 8-queens problem, Cryptarithmetic, Vacuum world, Missionaries and cannibals, simple route finding

- Real-world problem
  - More difficult
  - No single, agreed-upon specification (state, successor function, edge cost)
  - *Examples*: Route finding, VLSI layout, Robot navigation, Assembly sequencing

STATE SPACE REPRESENTATION

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

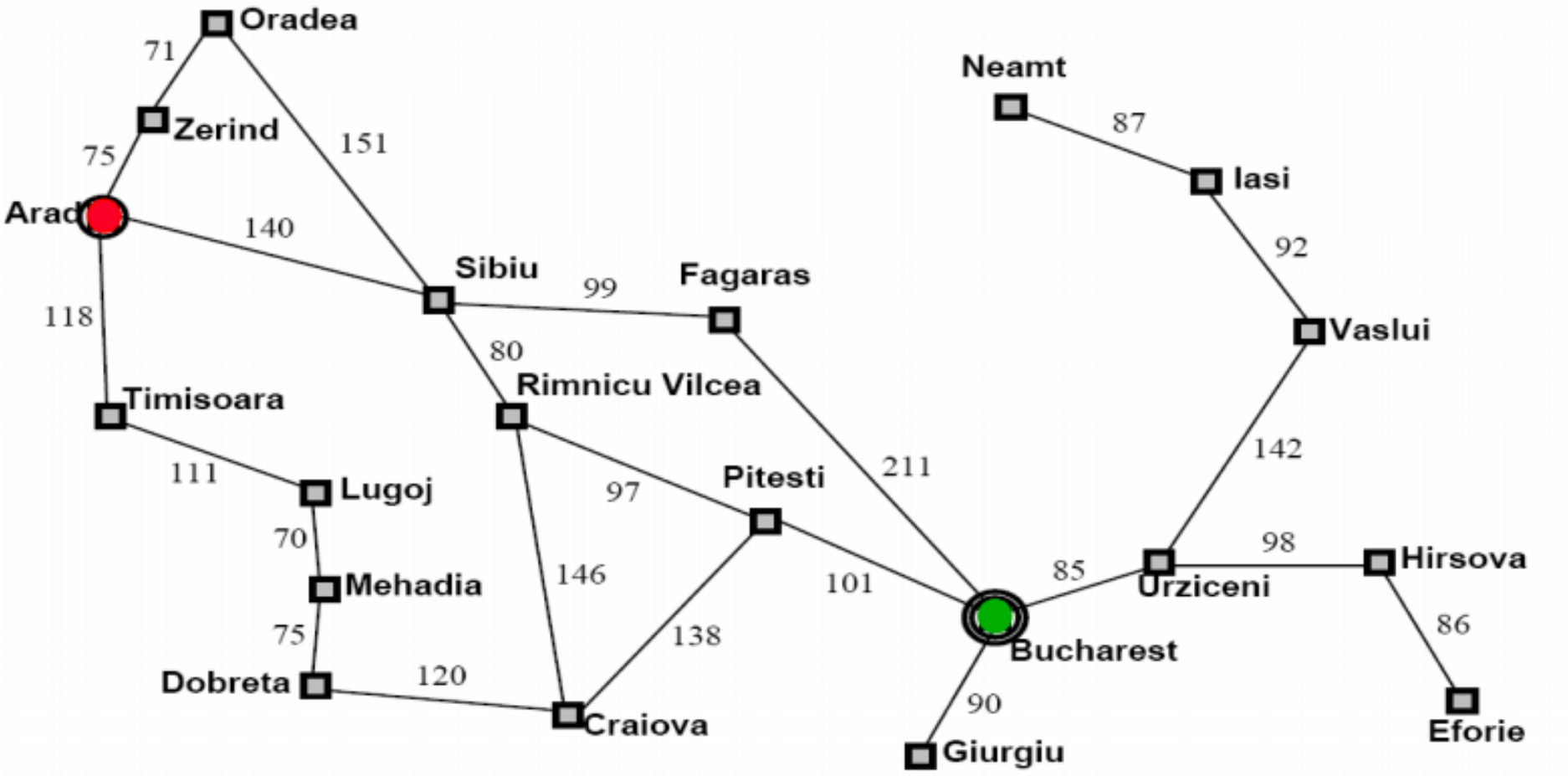
1. **INITIAL STATE** that the agent starts in.
2. Description of the **POSSIBLE ACTIONS** available to the agent.
3. **DESCRIPTION of what each action does**. [transition model]
4. **GOAL TEST**, which determines whether a given state is a goal state
5. **PATH COST FUNCTION** that assigns a numeric cost to each path

A SOLUTION TO A PROBLEM IS AN ACTION SEQUENCE THAT LEADS FROM THE INITIAL STATE TO A GOAL STATE.

Note: Solution quality = path cost function, & **OPTIMAL SOLUTION** → lowest path cost among all solutions

EXAMPLE 1

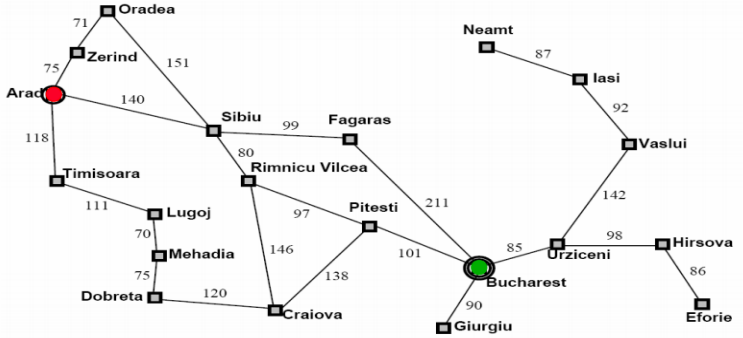
Currently in Arad, need to get to Bucharest by tomorrow to catch a flight. What is the State Space?



Example 1.

- State space.
  - States: the various cities you could be located in.
    - Note we are ignoring the low level details of driving, states where you are on the road between cities, etc.
  - Actions: drive between neighboring cities.
  - Initial state: in Arad
  - Desired condition (Goal): be in a state where you are in Bucharest. (How many states satisfy this condition?)
- Solution will be the route, the sequence of cities to travel through to get to Bucharest.

Problem Formulation –



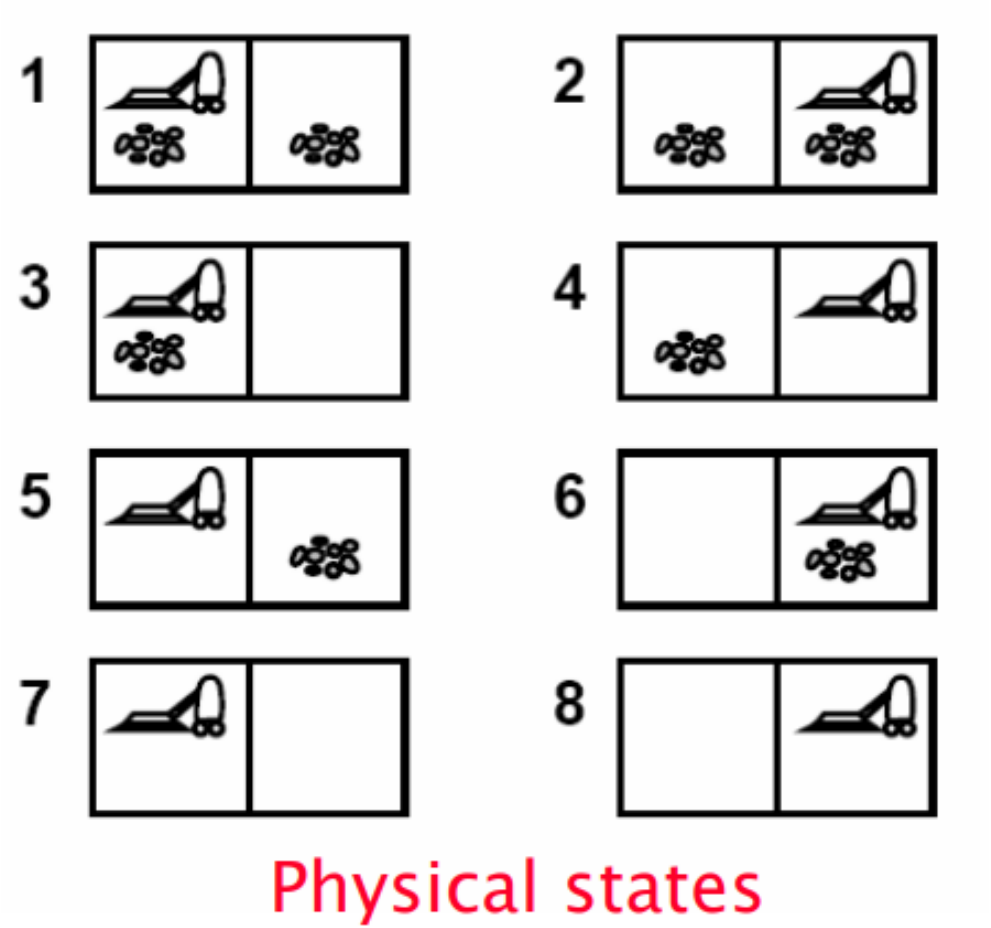
- Initial state: In(*Arad*)
- Actions, if current state is In(Arad), actions = {Go{Sibiu), Go(Timisoara), Go(Zerind)}
- Transition model:
  - e.g., Results(In(Arad), Go(*Sibiu*)) = In(*Sibiu*)
- Goal test determines whether a given state is a goal state
  - explicit, e.g. In(*Bucharest*)
  - implicit, e.g. checkmate
- Path cost function that assigns a numeric cost to each path
  - e.g., distance traveled
  - step cost: $c(x, a, y)$

- Solution: a path from the initial state to a goal state
- Optimal solution: the path that has the lowest path cost among all solutions; measured by the path cost function

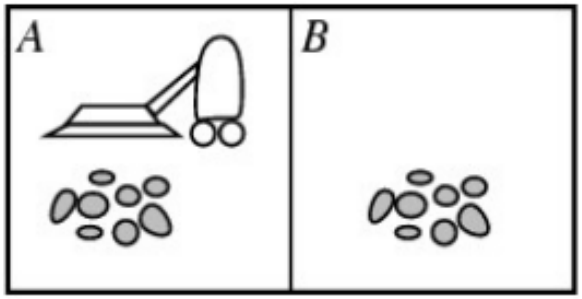
Example 2

Vacuum Cleaner

- The vacuum world
  - The world has only two *locations*
  - Each location may or may not contain *dirt*
  - The agent may be in one location or the other
  - 8 possible *world states*
  - Three possible actions: *Left, Right, Suction*
  - *Goal*: clean up all the dirt



Example: Vacuum-Cleaner



States

- 8 states

Initial state

- any state

Actions

- Left, Right, and Suck

Transition model

- complete state space, see next page

Goal test

- whether both squares are clean

Path cost

- each step costs 1

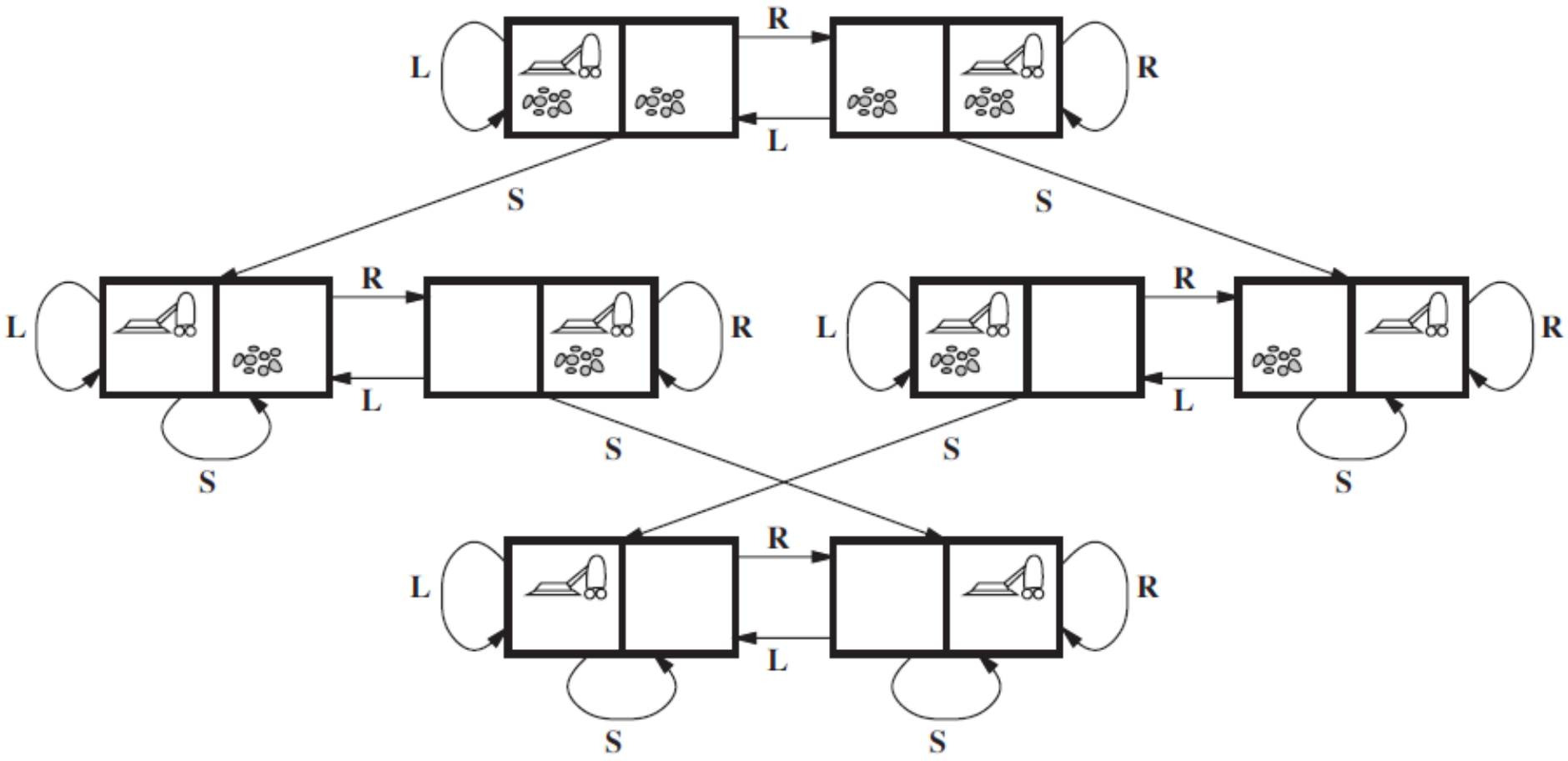


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Formulating a Vacuum Agent

- **States**: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.
- **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect. The complete state space is shown in Figure 3.3.
- **Goal test**: This checks whether all the squares are clean.
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

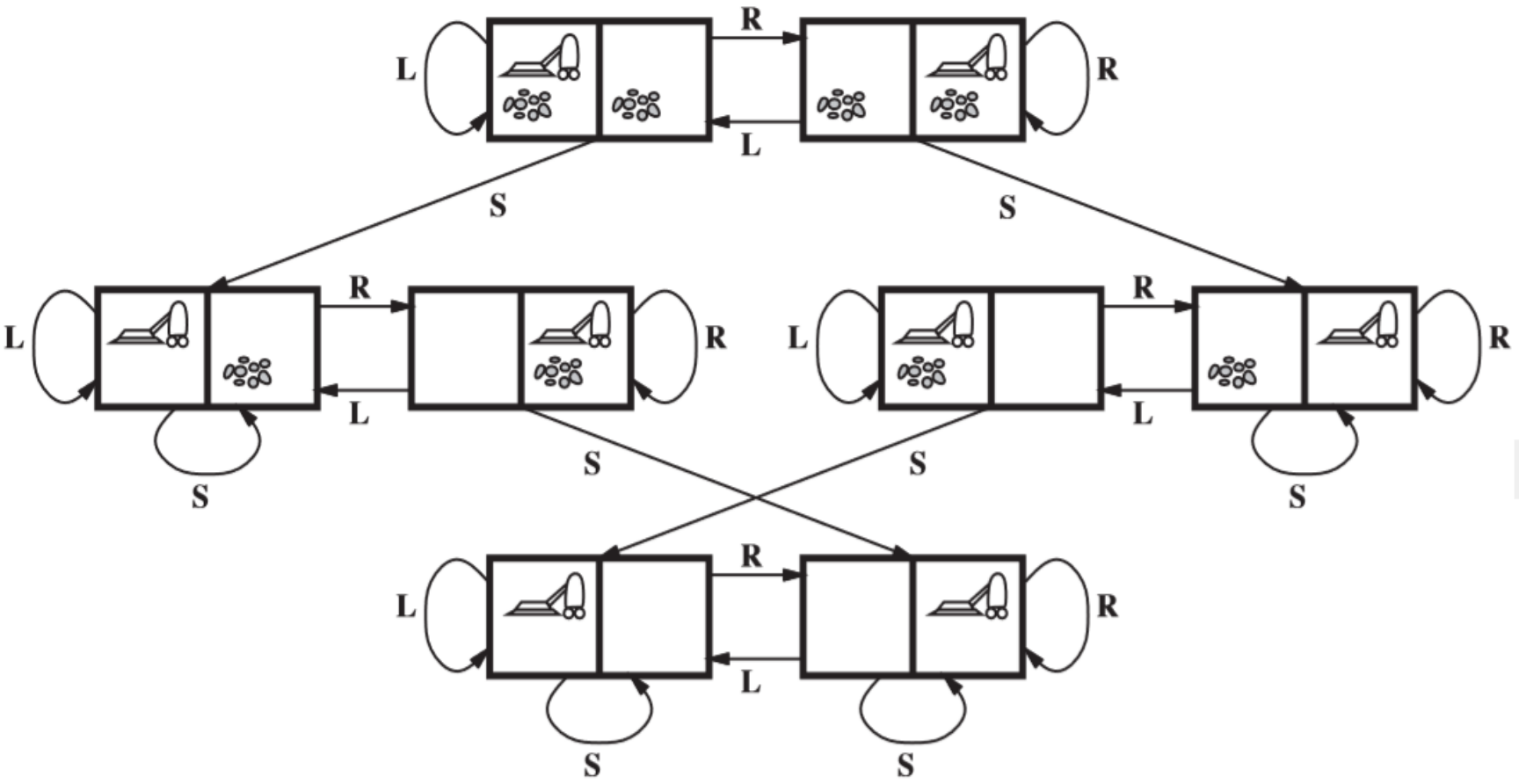


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

- Example 3

Missionaries and Cannibals

Question: In this problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, that the missionaries present on the bank cannot be outnumbered by cannibals. The boat cannot cross the river by itself with no people on board.

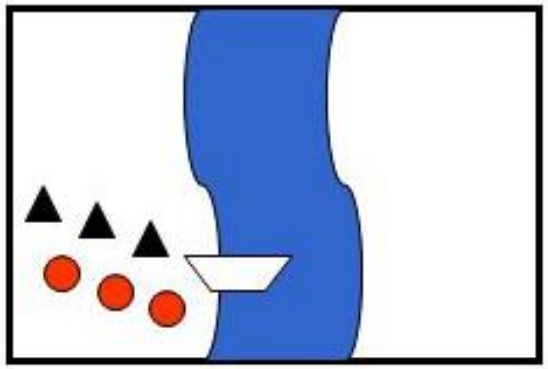
Short Video

Missionaries and cannibals

- There are three missionaries and three cannibals on the left bank of a river.
- They wish to cross over to the right bank using a boat that can only carry two at a time.
- The number of cannibals on either bank must never exceed the number of missionaries on the same bank, otherwise the missionaries will become the cannibals' dinner!
- Plan a sequence of crossings that will take everyone safely across.

Missionaries and Cannibals: Initial State and Actions

- initial state:
  - all missionaries, all cannibals, and the boat are on the left bank



- 5 possible actions:
  - one missionary crossing
  - one cannibal crossing
  - two missionaries crossing
  - two cannibals crossing
  - one missionary and one cannibal crossing

Solution

First let us consider that both the missionaries (M) and cannibals(C) are on the same side of the river- Left Right

Initially the positions are : 0M , 0C and 3M , 3C (B)

Now let's send 2 Cannibals to left of bank : 0M , 2C (B) and 3M , 1C

- Send one cannibal from left to right : 0M , 1C and 3M , 2C (B)
- Now send the 2 remaining Cannibals to left : 0M , 3C (B) and 3M , 0C
  Send 1 cannibal to the right : 0M , 2C and 3M , 1C (B)
- Now send 2 missionaries to the left : 2M , 2C (B) and 1M . 1C
- Send 1 missionary and 1 cannibal to right : 1M , 1C and 2M , 2C (B)
- Send 2 missionaries to left : 3M , 1C (B) and 0M , 2C
- Send 1 cannibal to right : 3M , 0C and 0M , 3C (B)
- Send 2 cannibals to left : 3M , 2C (B) and 0M , 1C
- Send 1 cannibal to right : 3M , 1C and 0M , 2C (B)'
- Send 2 cannibals to left : 3M , 3C (B) and 0M , 0C
- Here (B) shows the position of the boat after the action is performed. Therefore all the missionaries and cannibals have crossed the river safely.

Formulating Missionaries & Cannibal Problem

- **State space**: Triple (**x** , **y** , **z**) with 0 ≤ x , y , z ≤ 3, where x , y , and z represent the number of **missionaries**, **cannibals** and **boats** currently on the original bank.
- **Initial State**: (3 , 3 , 1)
  - Successor function: From each state, either bring
    - one missionary, one cannibal,
    - two missionaries, two cannibals, or
    - one of each type to the other bank.
    - Note: Not all states are attainable (e.g., (0,0,1)), and some are illegal.
- Goal State: (0,0,0)
- Path Costs: 1 unit per crossing

Example 4

The 8 Puzzle

Sliding-block/tile-puzzle(most popular instruments in AI studies)

[toy-problem]

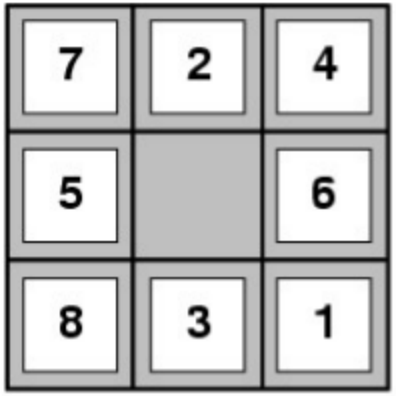
- Arrange the tiles so that all the tiles are in the correct positions.
  - You can move the BLANK tile up, down, left, or right, so long as the following conditions are met:
- There's no other tile blocking you in the direction of the movement;
- You are not trying to move outside of the boundaries/edges.

INITIAL	GOAL
1 2 3	1 2 3
* 4 6	4 5 6
7 5 8	7 * 8

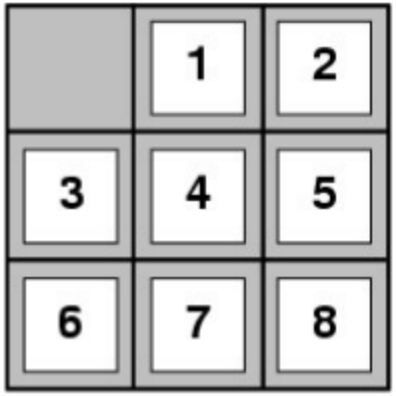
Formulating for 8 puzzle problem

- **States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states (Exercise 3.4).
- **Actions**: The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
- **Transition model**: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
- **Goal test**: This checks whether the state matches the goal configuration shown in Figure 3.4. (Other goal configurations are possible.)
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

Example: 8-puzzle



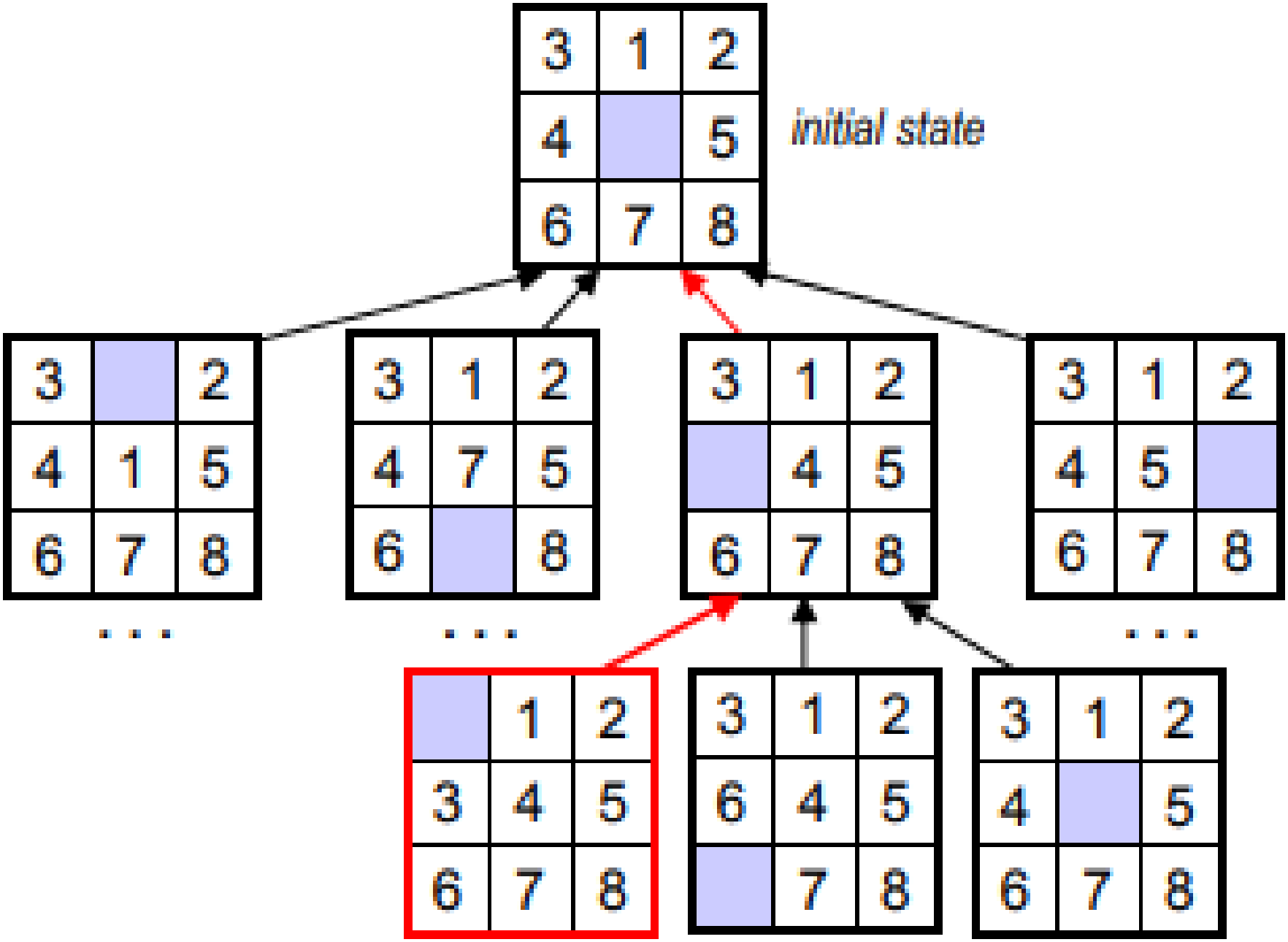
Start State

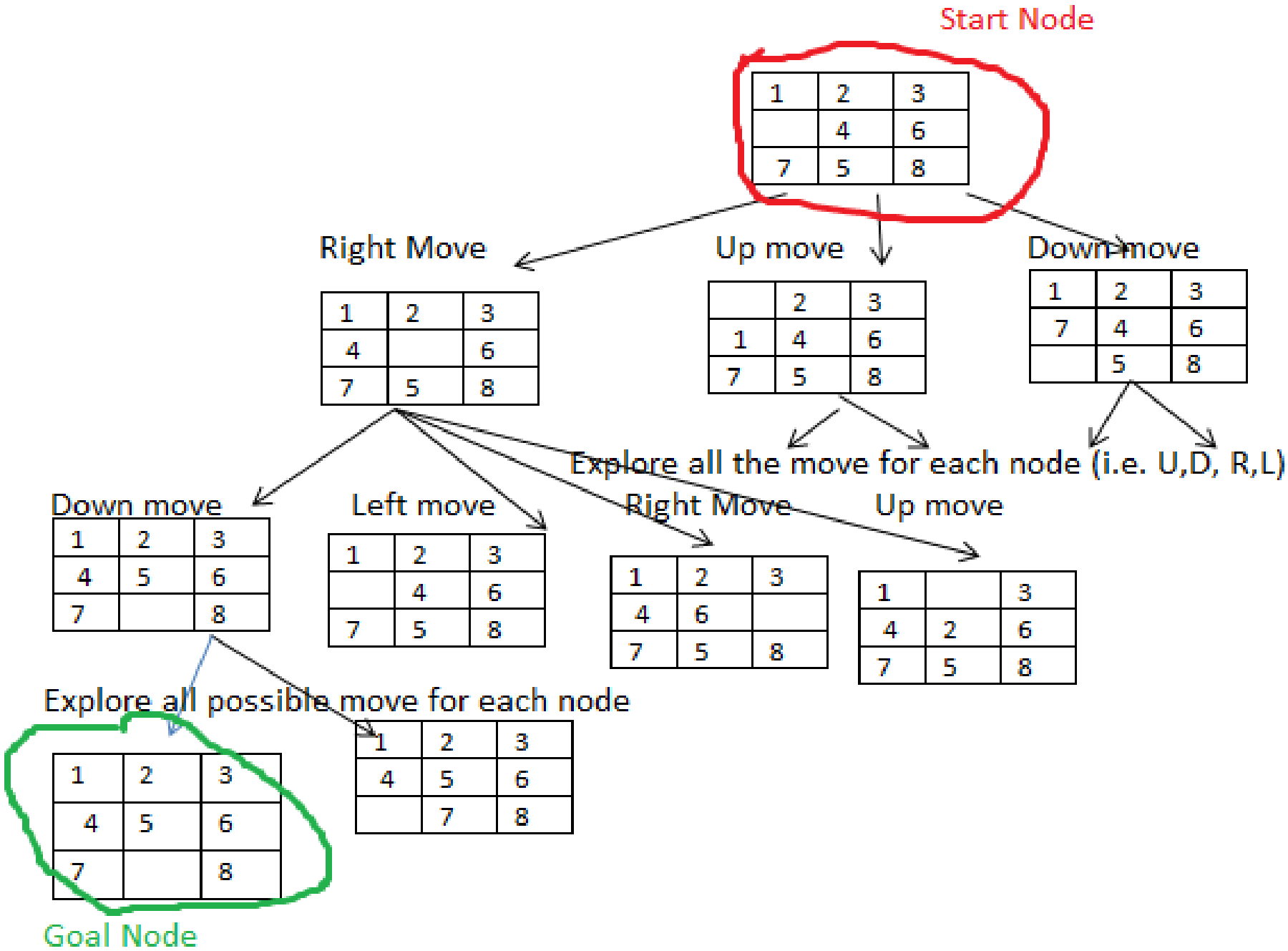


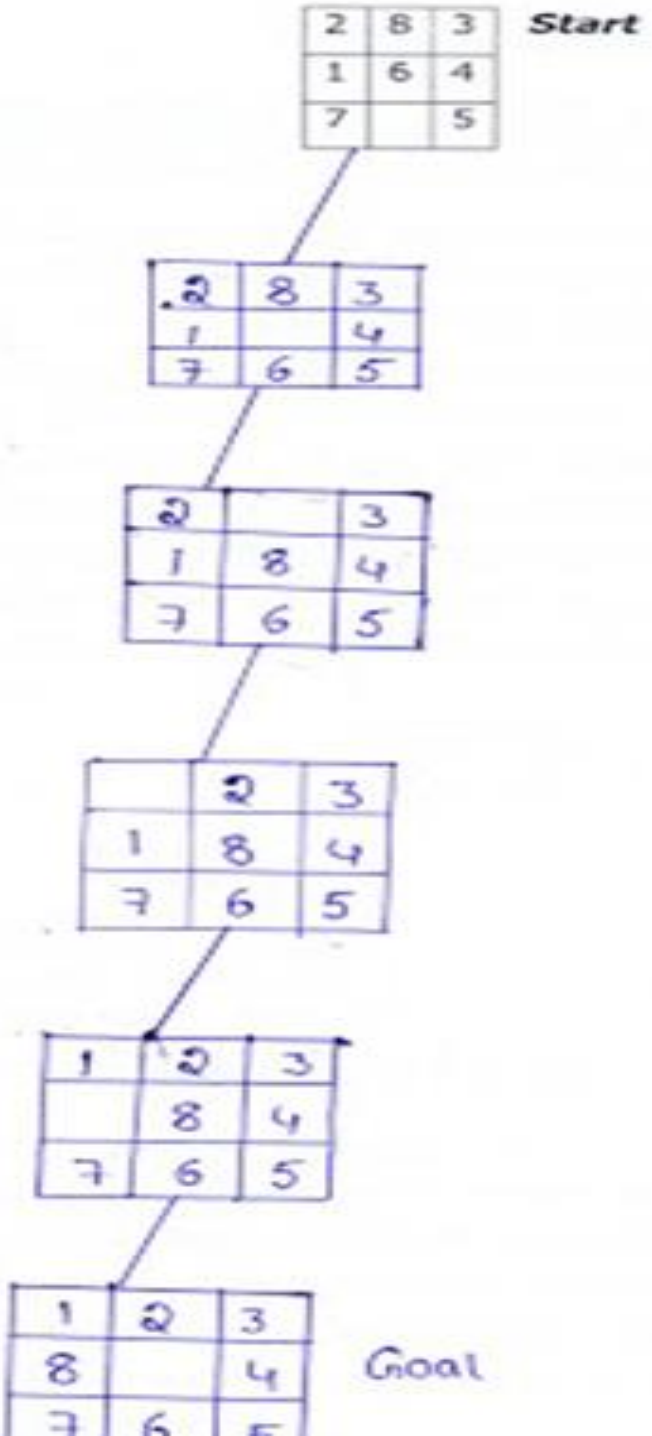
Goal State

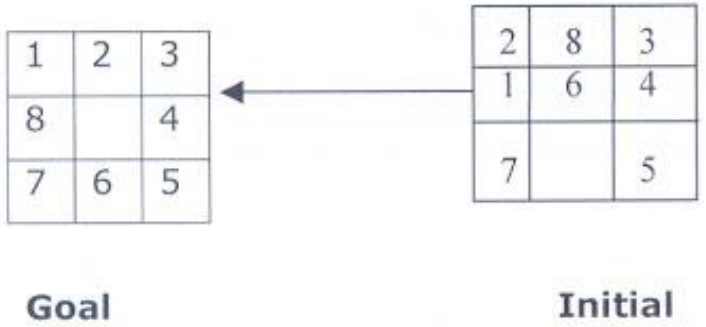
NP-Complete

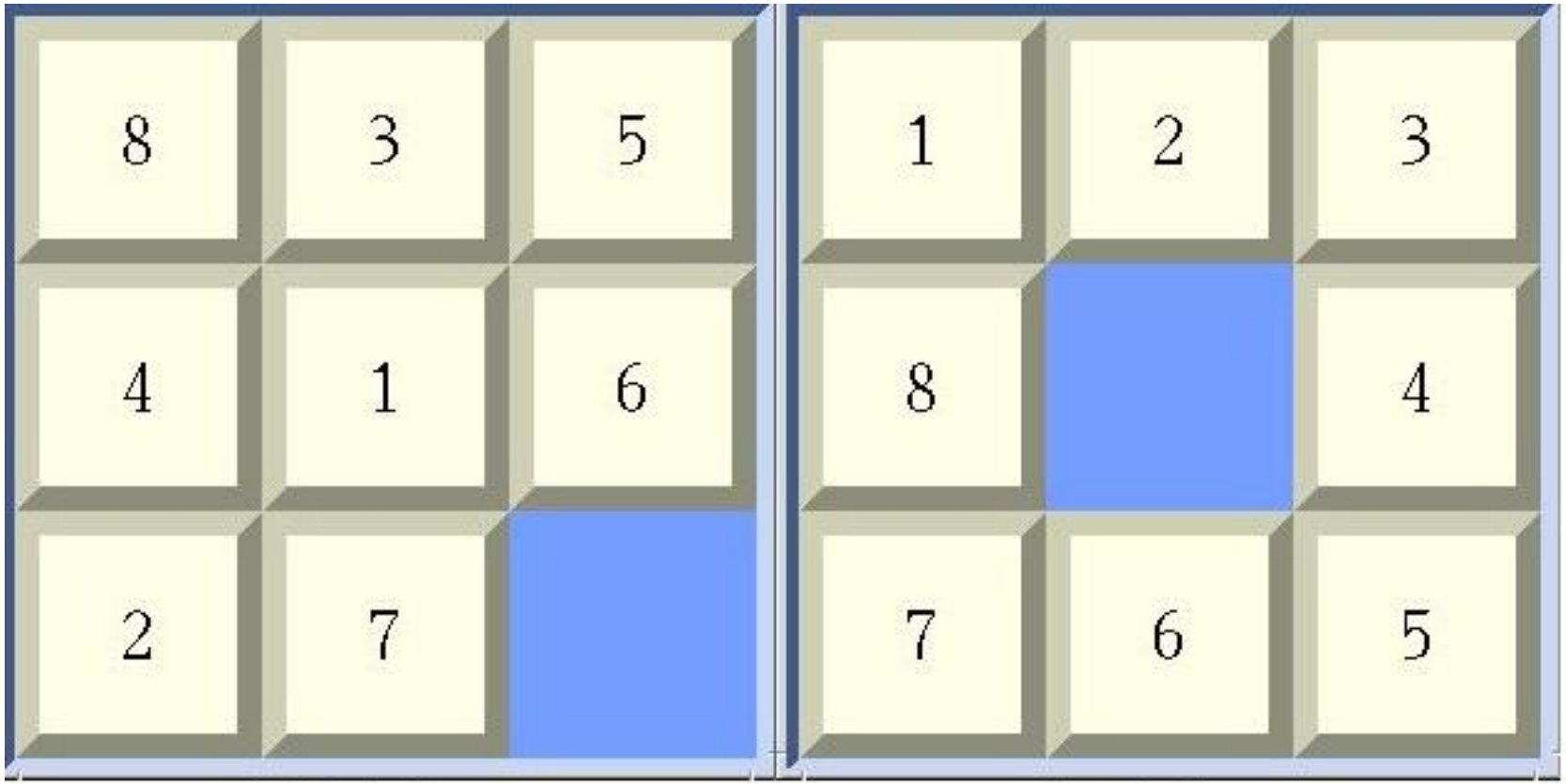
- States:
  - location of each tile and the blank
- Initial state: any, 9!/2
- Actions:
  - blank moves Left, Right, Up or Down
- Transition model:
  - Given a state and action, returns the resulting state
- Goal test: Goal configuration
- Path cost: Each step costs 1


3 1 2
4 5
6 7 8
initial state
3 2
4 1 5
6 7 8
3 1 2
4 7 5
6 8
3 1 2
4 5
6 7 8
3 1 2
4 5
6 7 8
1 2
3 4 5
6 7 8
3 1 2
6 4 5
7 8
3 1 2
4 5
6 7 8


Start Node
1 2 3
4 6
7 5 8
Right Move
1 2 3
4 6
7 5 8
Up move
2 3
1 4 6
7 5 8
Down move
1 2 3
7 4 6
5 8
Explore all the move for each node (i.e. U,D, R,L)
Down move
1 2 3
4 5 6
7 8
Left move
1 2 3
4 6
7 5 8
Right Move
1 2 3
4 6
7 5 8
Up move
1 3
4 2 6
7 5 8
Explore all possible move for each node
1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8
Goal Node


Start
Goal

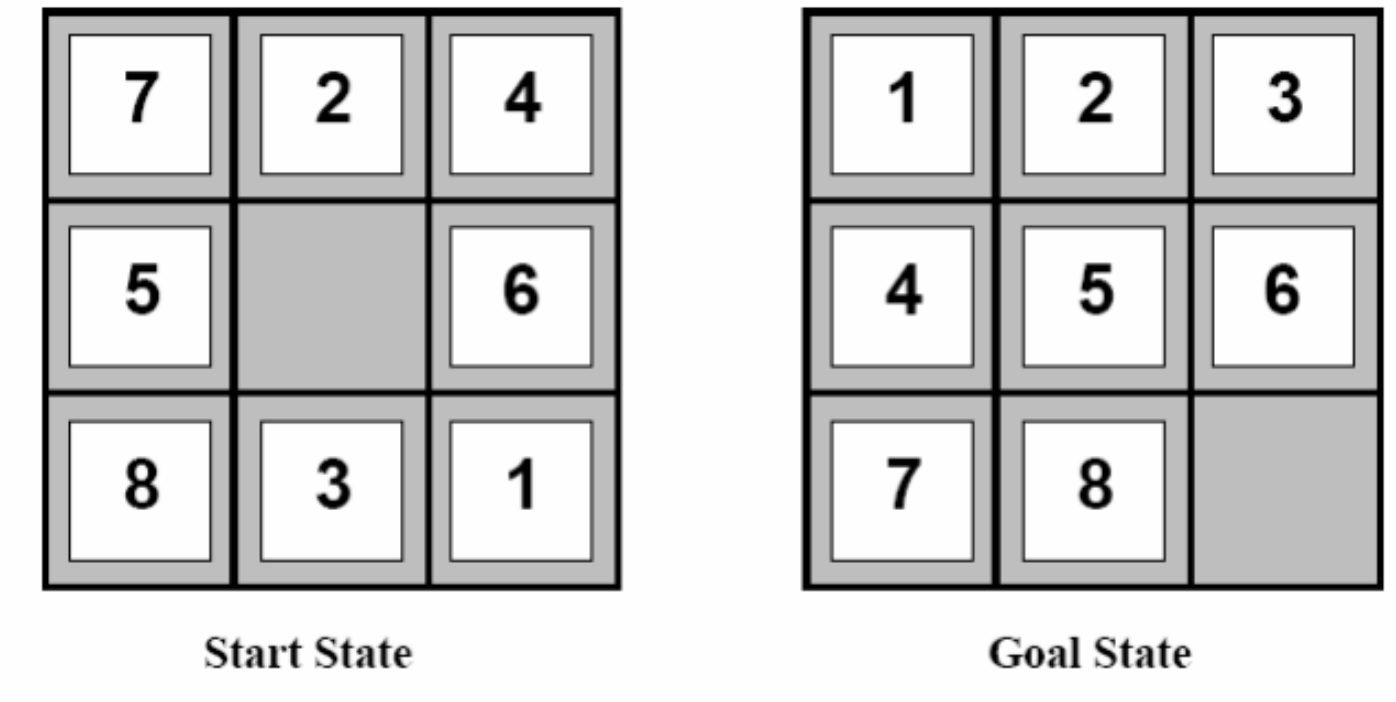

Goal
Initial



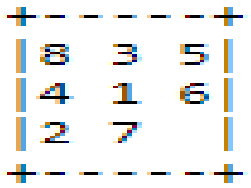
Initial State

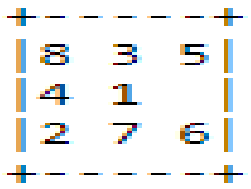
Goal State

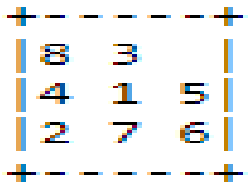
8-puzzle

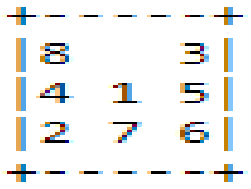


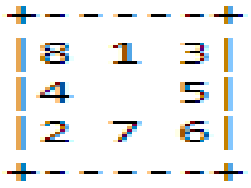
Rule: Can slide a tile into the blank spot. (Equivalently, can think if it as moving the blank around).

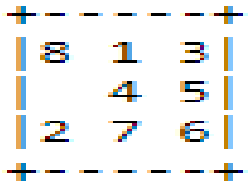

8 3 5
4 1 6
2 7

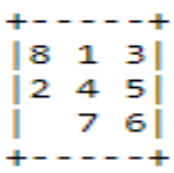

8 3 5
4 1
2 7 6

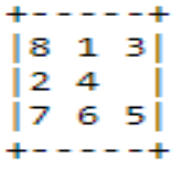

8 3
4 1 5
2 7 6

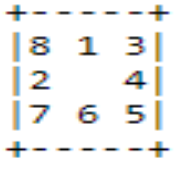

8 3
4 1 5
2 7 6

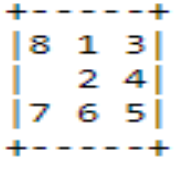

8 1 3
4 5
2 7 6

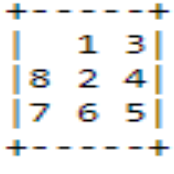

8 1 3
4 5
2 7 6

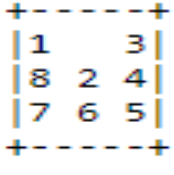

8 1 3
2 4 5
7 6

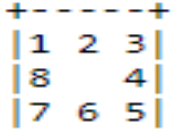

8 1 3
2 4
7 6 5


8 1 3
2 4
7 6 5

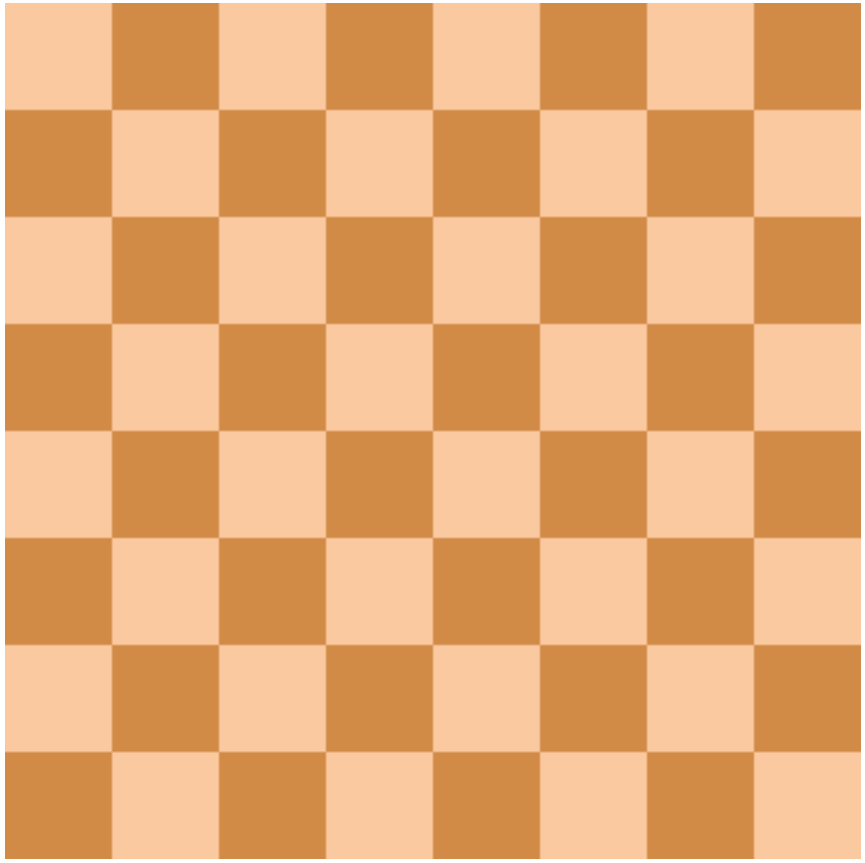

8 1 3
2 4
7 6 5

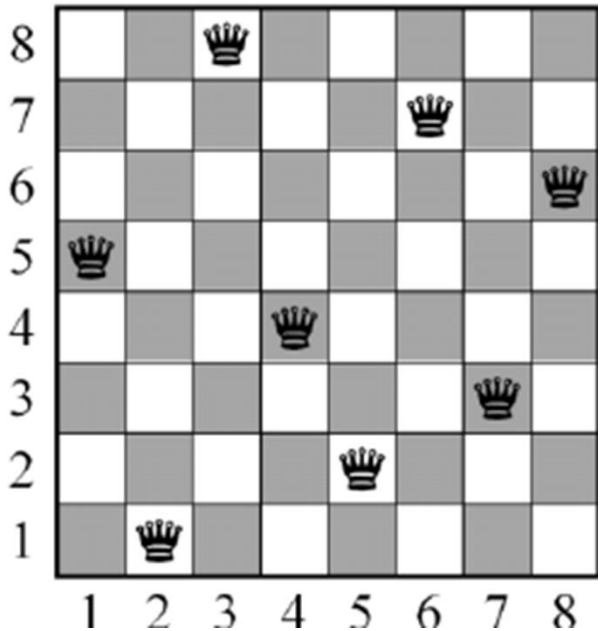

1 3
8 2 4
7 6 5


1 3
8 2 4
7 6 5


1 2 3
8 4
7 6 5

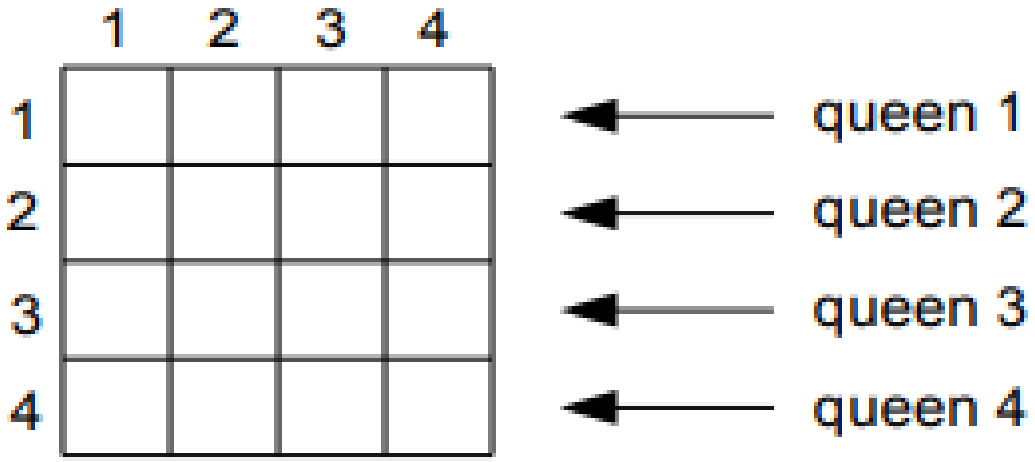
- Example 5





4 Queen Problem

THE N-QUEEN PROBLEM: Place n queens on an n by n chess board so that no two of them are on the same row, column, or diagonal .



8-queens problem

The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other.

(A queen attacks any piece in the same row, column or diagonal.)

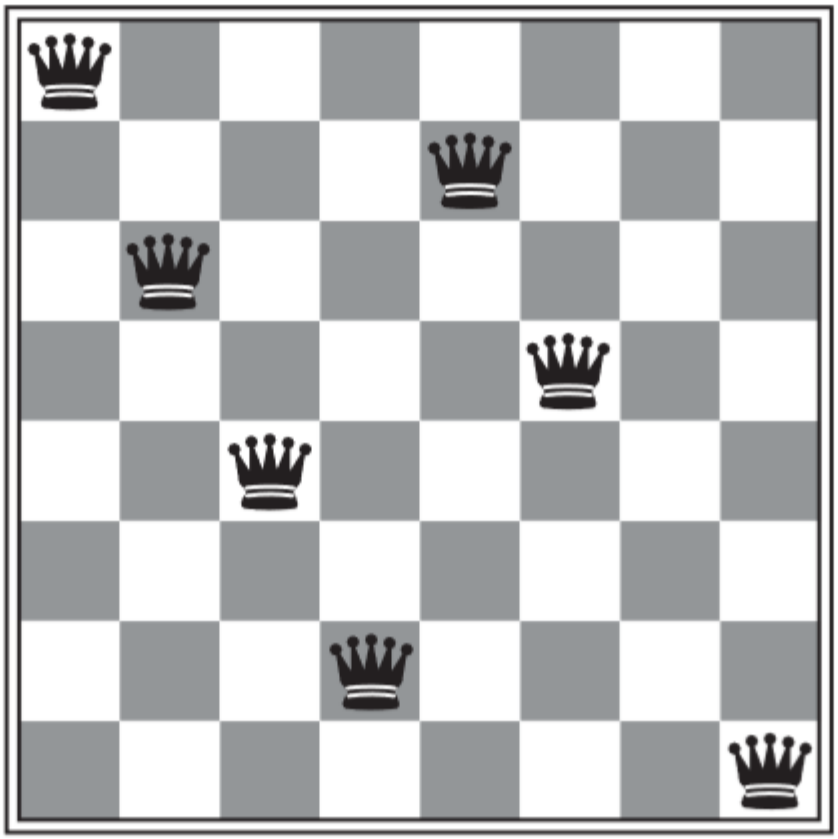
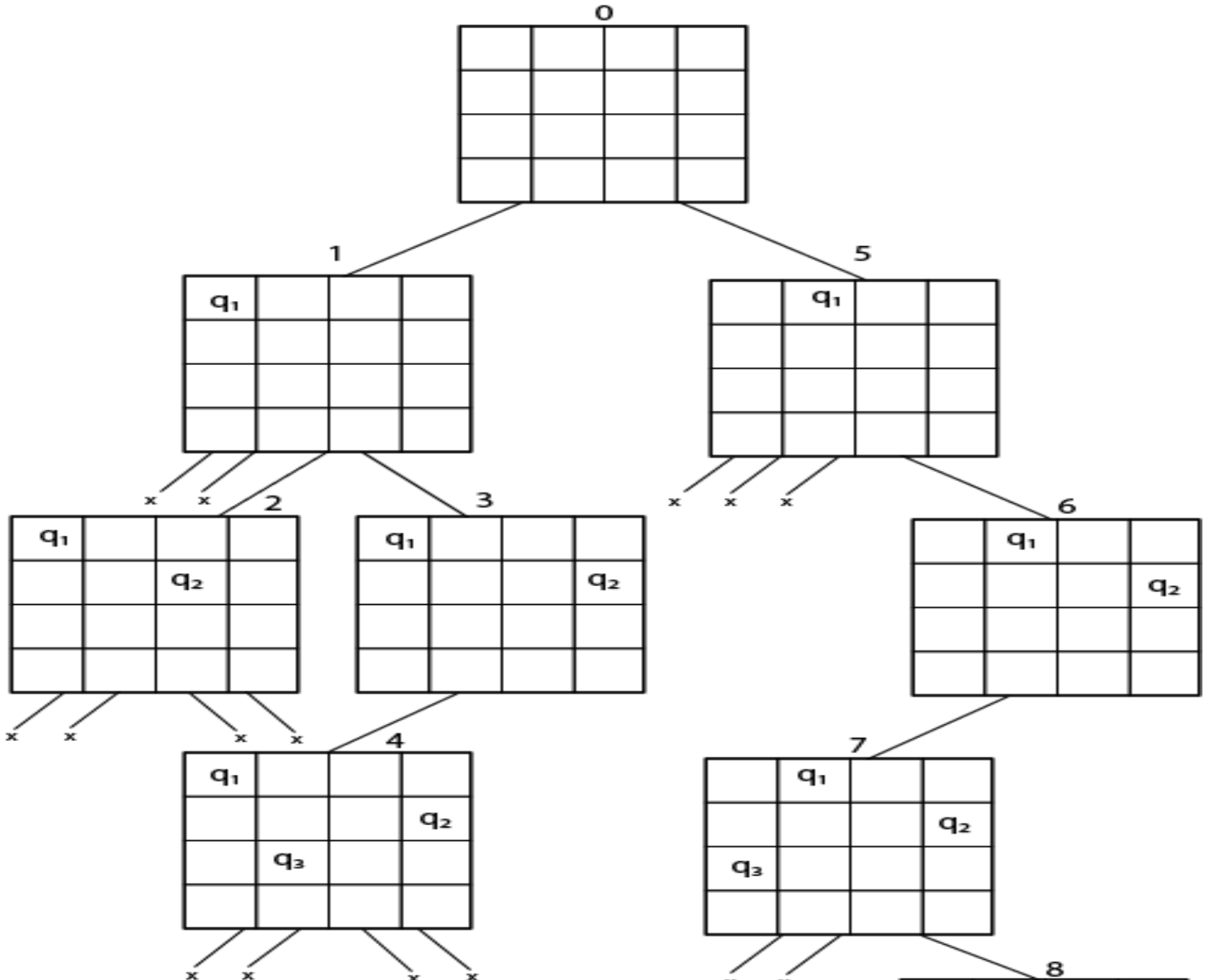


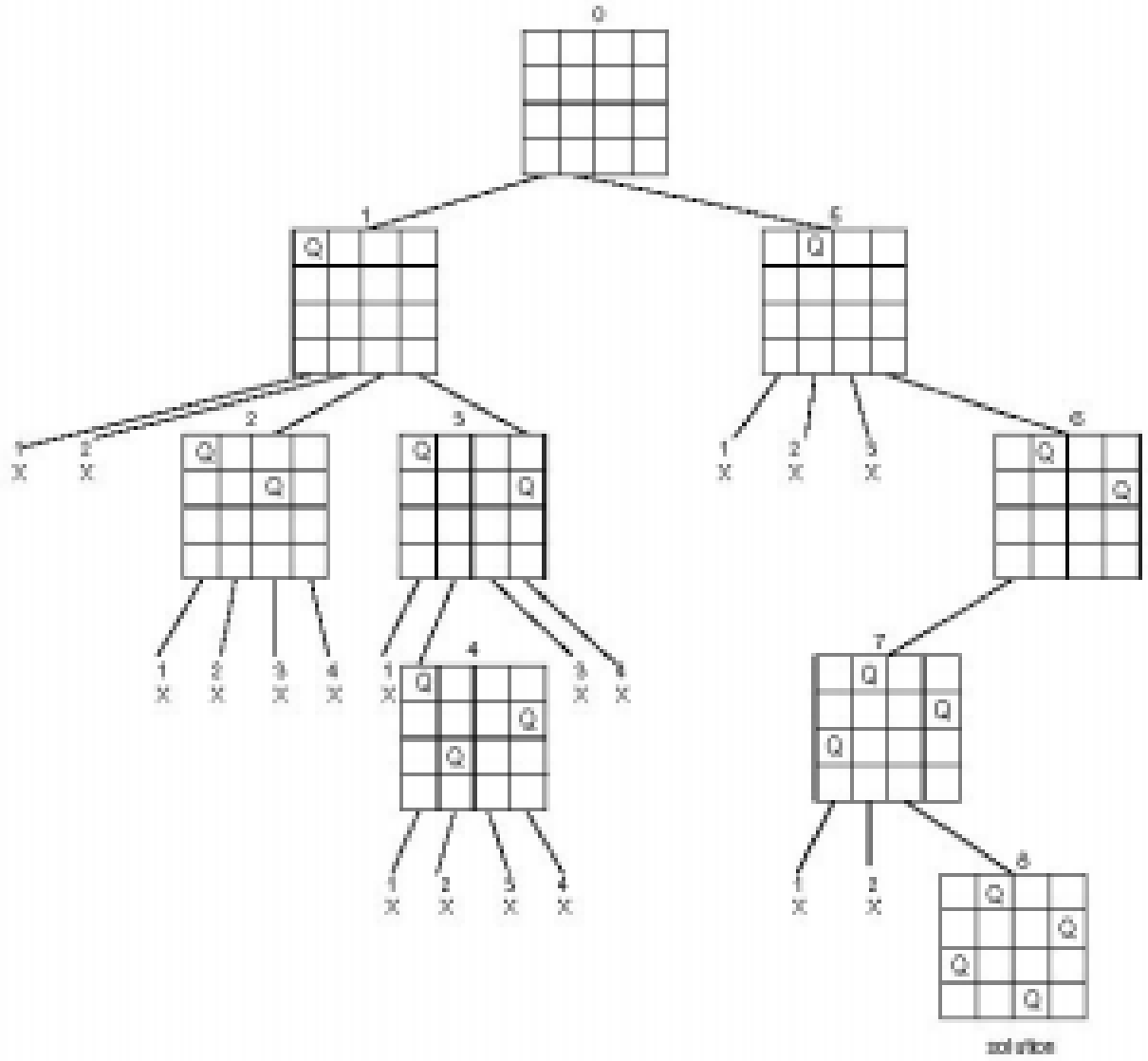
Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

Formulating 8 Queens Problem

- **States**: Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state**: No queens on the board.
- **Actions**: Add a queen to any empty square.
- **Transition model**: Returns the board with a queen added to the specified square.
- **Goal test**: 8 queens are on the board, none attacked.

Draw the State Space tree for 4 Queen Problem


0
1
5
q1
q1
x
x
x
x
x
2
3
6
q1
q2
q1
q2
q1
q2
x
x
x
x
4
7
q1
q2
q3
q1
q2
q3
8


0
1
5
2
3
6
7
8
solution

4 Queen

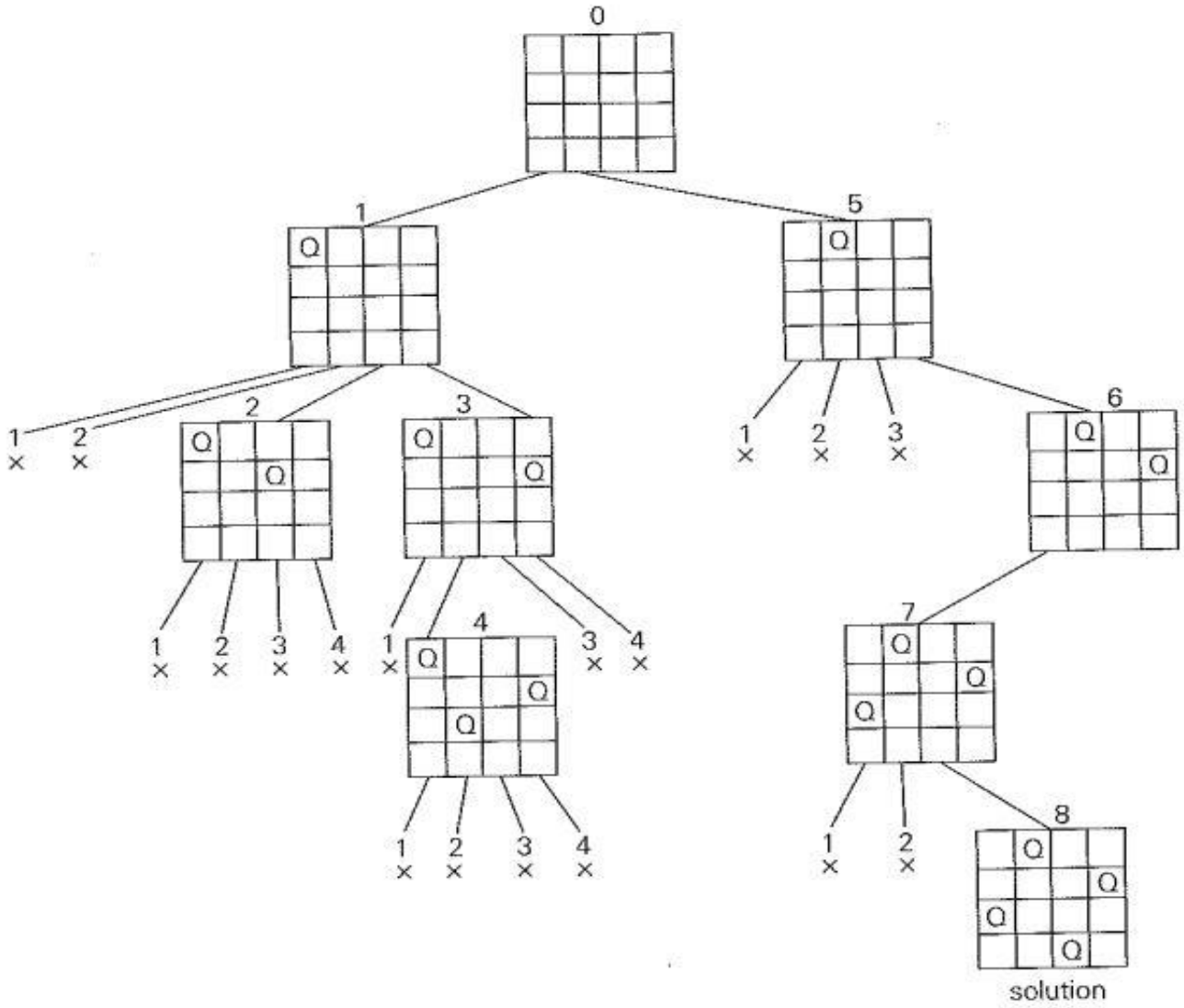
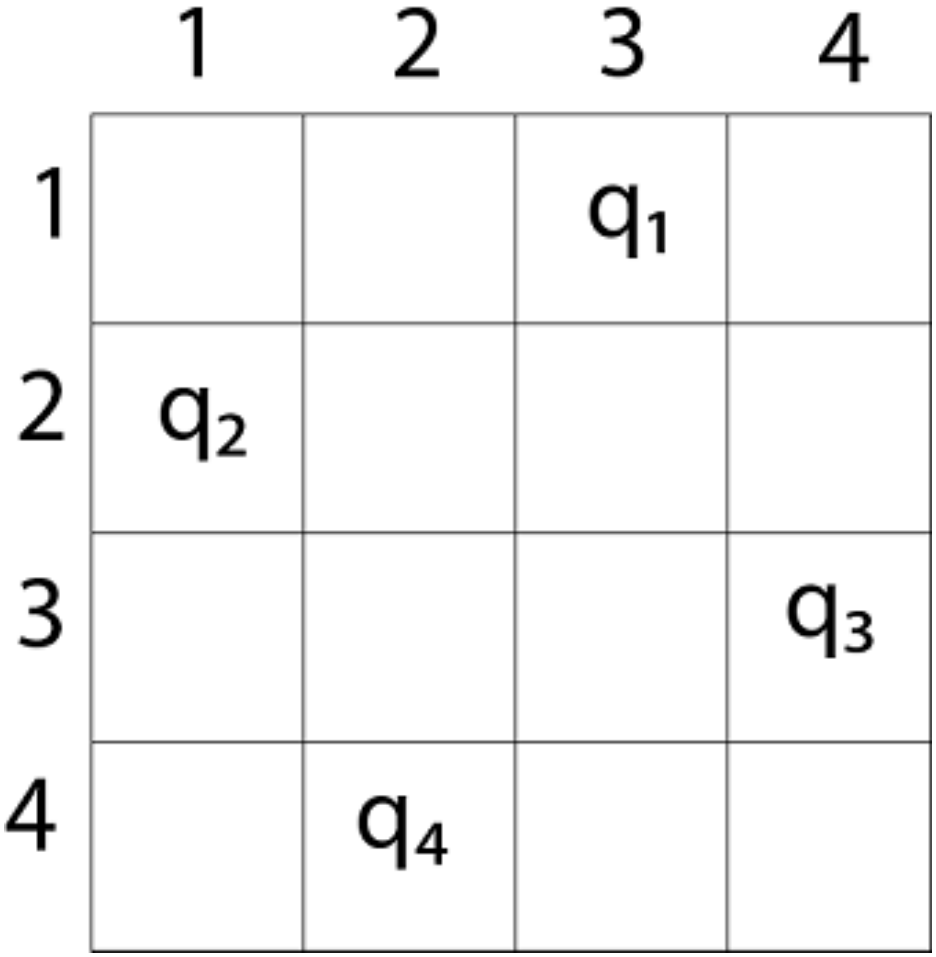


FIGURE 12.2 State-space tree of solving the four-queens problem by backtracking. × denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

Possible Solution



Example 6

Water Jug Problem

4 L

3 L

Water Jug Problem

- **Problem:** You are given two jugs, a 4-gallon one and a 3-gallon one . Neither has any measuring mark on it . There is a pump that can be used to fill the jugs with water . How can you get exactly 2 gallons of water into the 4-gallon jug.
-

1.	Fill the 4-gallon jug
2.	Fill the 3-gallon jug
3.	Pour some water out of the 4-gallon jug
4.	Pour some water out of 3-gallon jug
5.	Empty the 4-gallon jug on the ground
6.	Empty the 3-gallon jug on the ground
7.	Pour water from the 3-gallon jug into the 4-gallon jug until the gallon jug is full.
8.	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9.	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	Pour the 2-gallons water from 3-gallon jug into the 4;gallon jug
12.	Empty the 2-gallons in the 4-gallon jug on the ground.

X is 4L and Y is 3L

1. X = 0 , Y = 3 .
2. X = 3 , Y = 0 . (Transfer Y to X)
3. X = 3 , Y = 3 .(Capacity of X being 4)
4. X =4 , Y = 2 . (Fill X up to brim)
5. X=2 , Y= 0 . (Fill X with contents of Y after emptying X)

REAL WORLD PROBLEMS

- Web sites and in-car systems that provide driving directions
- Touring Problems
- Travelling Salesman Problem
- Robot Navigation
- Others, such as
  - routing video streams in computer networks
  - military operations planning
  - airline travel planning systems

CONSIDER THE AIRLINE TRAVEL PROBLEMS THAT MUST BE SOLVED BY A TRAVEL-PLANNING WEB SITE

- **States**: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
- **Initial state**: This is specified by the user's query.
- **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model**: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
- **Goal test**: Are we at the final destination specified by the user?
- **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

- **A VLSI layout problem** requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.
- **PROTEIN DESIGN** in which the goal is to find a sequence of amino acids that will fold into a three-dimensional protein with the right properties to cure some disease.

Defining Problem as a **STATE SPACE SEARCH**

1. A state space that contains **all possible configurations** of all relevant objects.
2. Specify one or more states describing all possible situations from which problem solving may start **(Initial States)**
3. Specify one or more states that would be acceptable as solutions to problems **(Goal States).**
4. Specify set of rules that describe **actions** (operators) available.
    1. What unstated assumptions are present in the informal problem description?
    2. How general should the rules be ?
    3. How much of the work required to solve the problem should be pre computed and represented in the rules?

Production Systems

- A set of **rules**(Left side[pattern] determining applicability of rule on right side[describes operation to be performed if rule applied])
- One or more **knowledge base** (Structured)
- **Control strategy**-order of rules applied/compared , way of resolving conflicts.
- **A rule applier**

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the fou gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],3)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon j

8 puzzle problem

Start state:

2	8	3
1	6	4
7		5

Goal state:

1	2	3
8		4
7	6	5

Production set:

Condition		Action
goal state in working memory	→	halt
blank is not on the left edge	→	move the blank left
blank is not on the top edge	→	move the blank up
blank is not on the right edge	→	move the blank right
blank is not on the bottomedge	→	move the blank down

Working memory is the present board state and goal state.

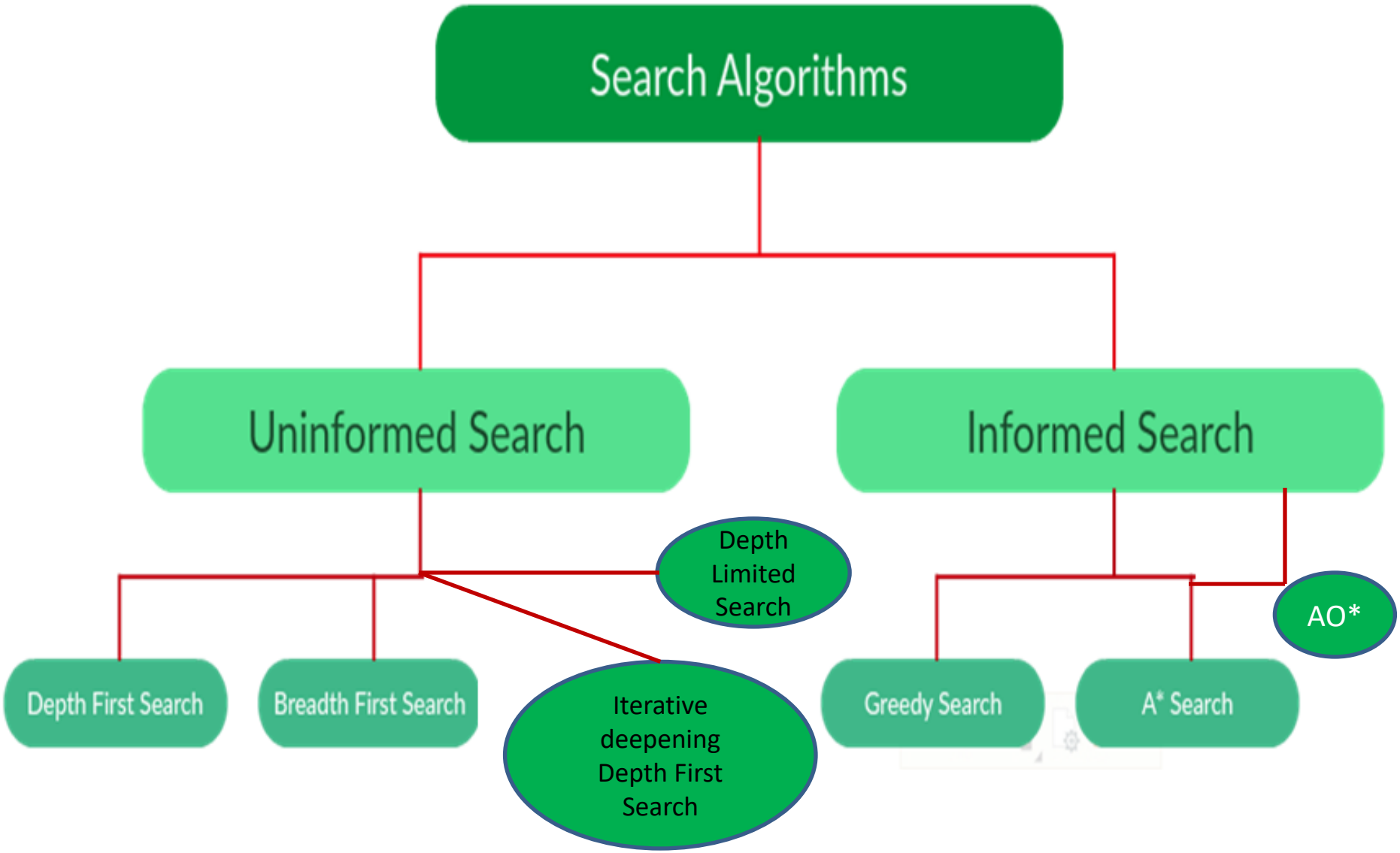
Problem Characteristics

1. **Is the problem decomposable?** Sub goal
2. **Can solution steps be ignored or undone ?**Eg (8 puzzle ,Chess , Theorem Proving)
    1. Ignorable , Recoverable , Irrecoverable (3 classes of problems).
3. **Is the universe predictable?**(Plan Problem Solving?) Eg Lawyer,Controlling Robot arm…
4. **Is a Good solution Absolute or relative?** Good Heuristics play imp role.
5. **Is the Solution a state or a path?**
6. **What is the role of knowledge?** Eg Chess-Strategy and Tactics, Newspaper reading?
7. **Does the task require interaction with a person?** Solitary and Conversational. Eg Medical diagnosis v/s peoples reaction?
8. **Problem Classification**

Issues in designing of search problems

- **Direction of search**(Forwards v/s Backward reasoning)
- **Selecting matching** (applicable)r**ules.**
  - Efficient procedures to match rules against states in production system.
- **Knowledge Representational Problem**-representing each node of the search process-Eg Chess (Array[small] v/s ??)

Searching for Solutions

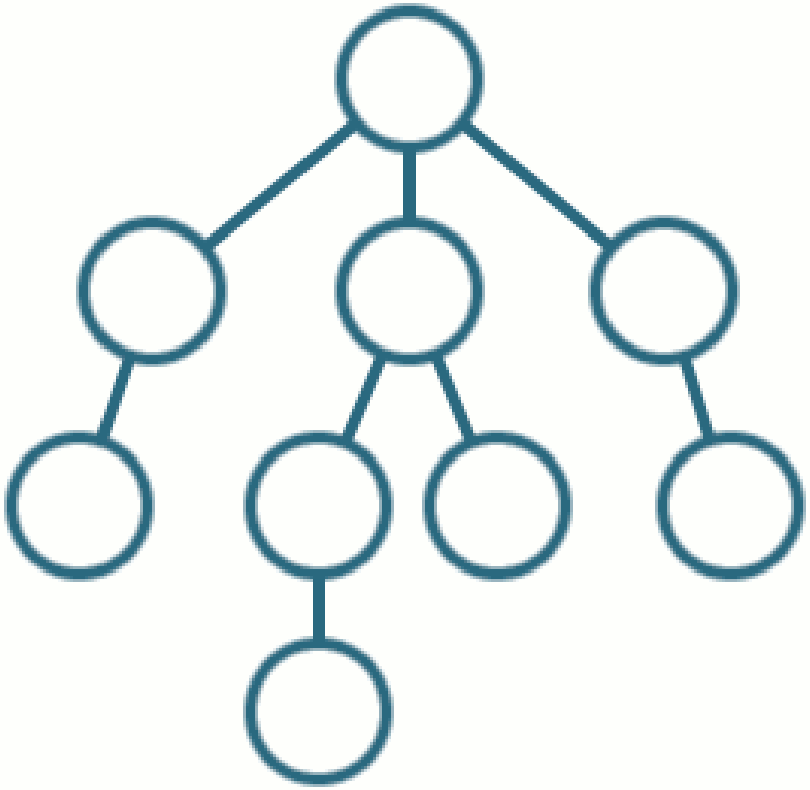

Search Algorithms
Uninformed Search
Informed Search
Depth Limited Search
AO*
Depth First Search
Breadth First Search
Iterative deepening Depth First Search
Greedy Search
A* Search

UNINFORMED SEARCH STRATEGIES(blind search)

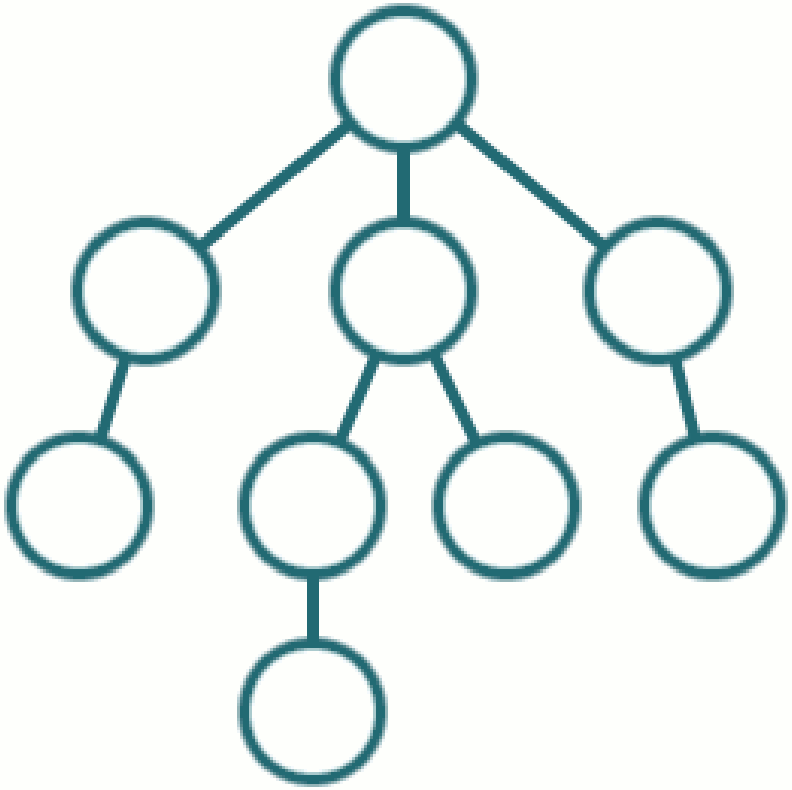
"Strategies have **no additional information** about states beyond that provided in the **problem definition**"

- All they can do is generate successors and **distinguish a goal state from a non-goal state.**
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Depth Limited search(DLS)
  - Iterative Deepening depth first search(IDDS)

DFS



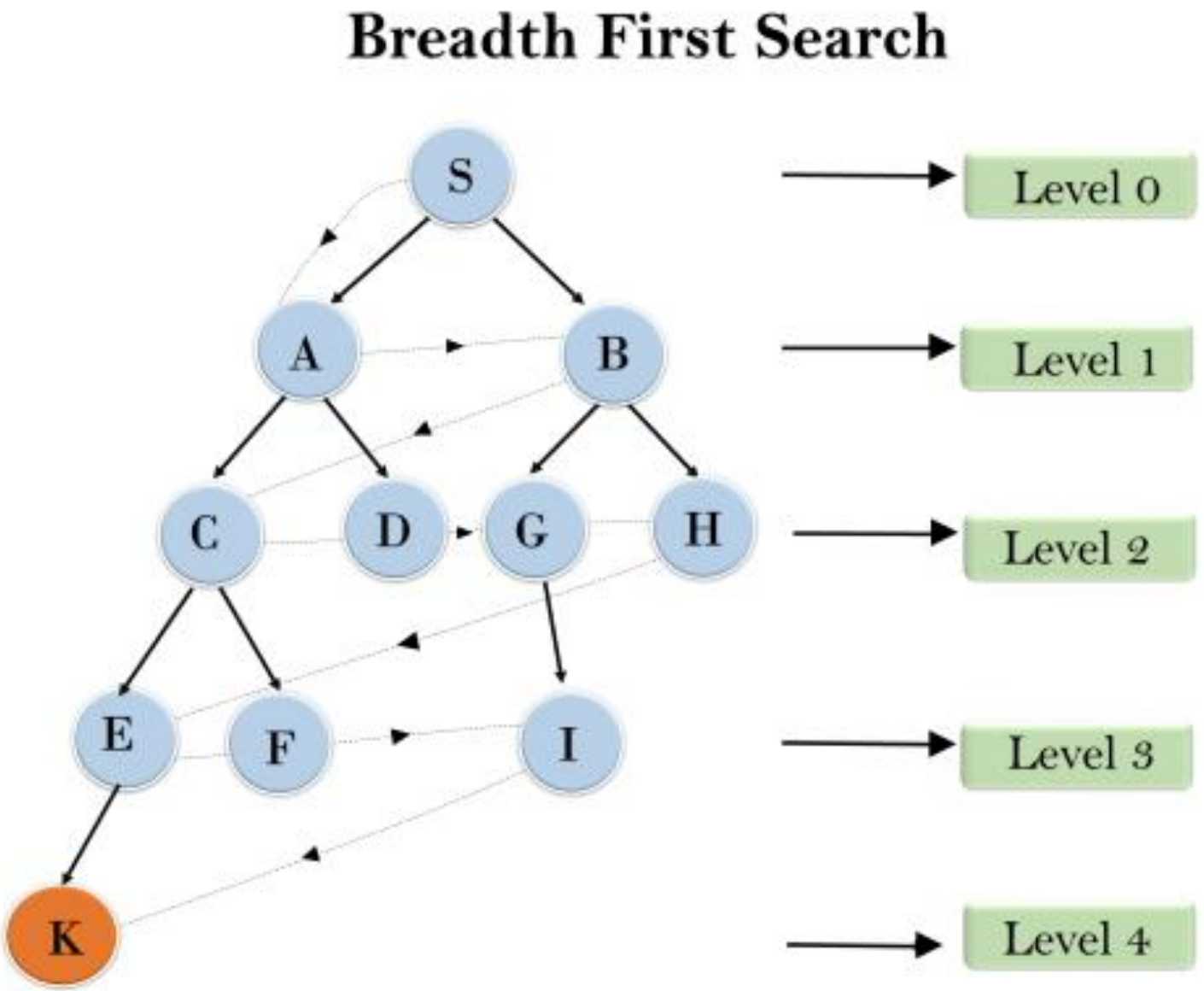
BFS



BREADTH FIRST SEARCH

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.

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



ADVANTAGES:

- BFS will **provide a solution** if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the **minimal solution** which requires the least number of steps.

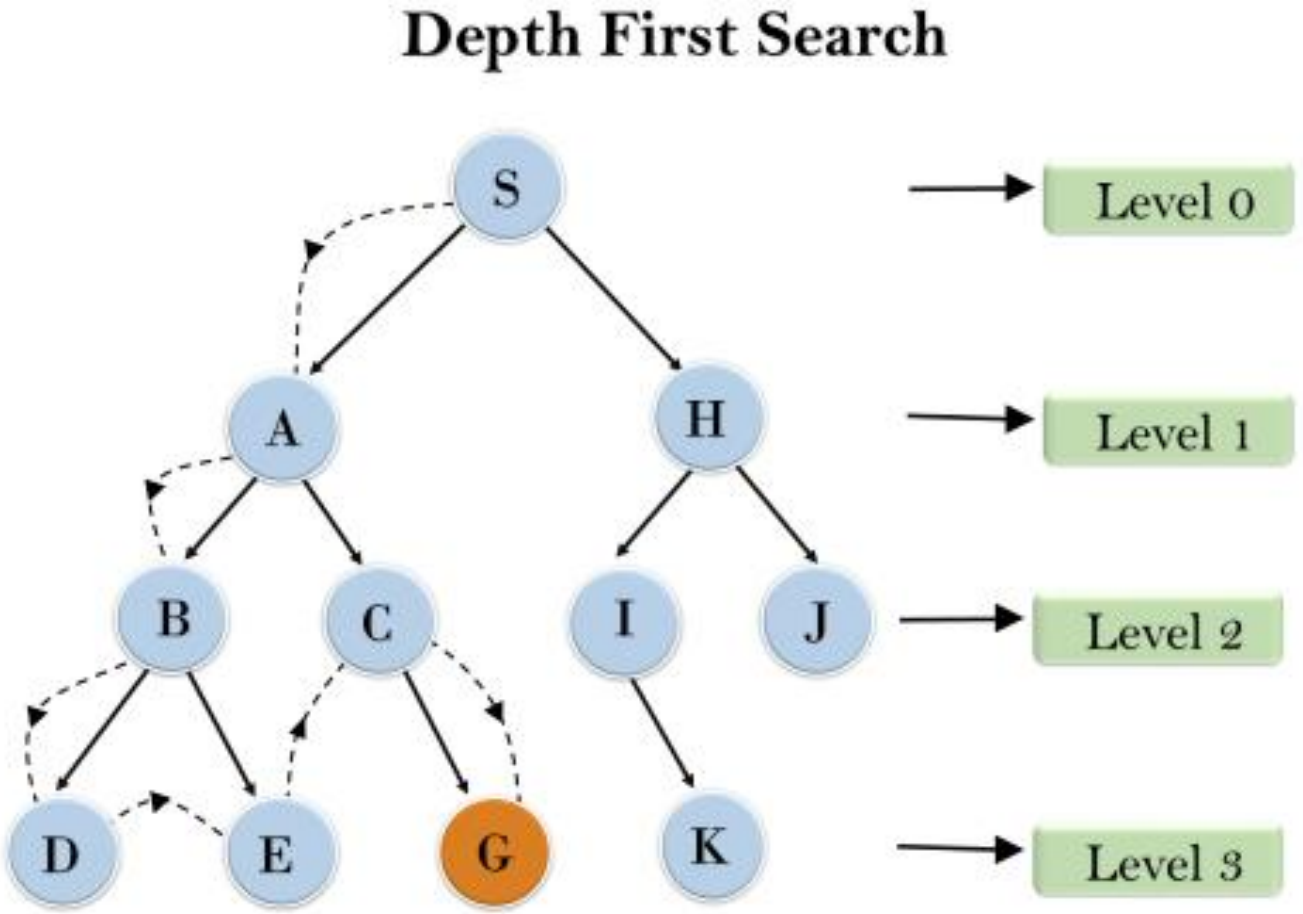
DISADVANTAGES:

- It requires **lots of memory** since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is **far away from the root node.**

DEPTH-FIRST SEARCH

- Depth-first search always expands the deepest node in the current frontier of the search tree.

Root node--->Left node ----> right node.



Advantages:

- DFS consumes very less memory space.

- It will reach at the goal node in a less time period than BFS if it traverses in a right path.

- It may find a solution without examining much of search because we may get the desired solution in the very first go.

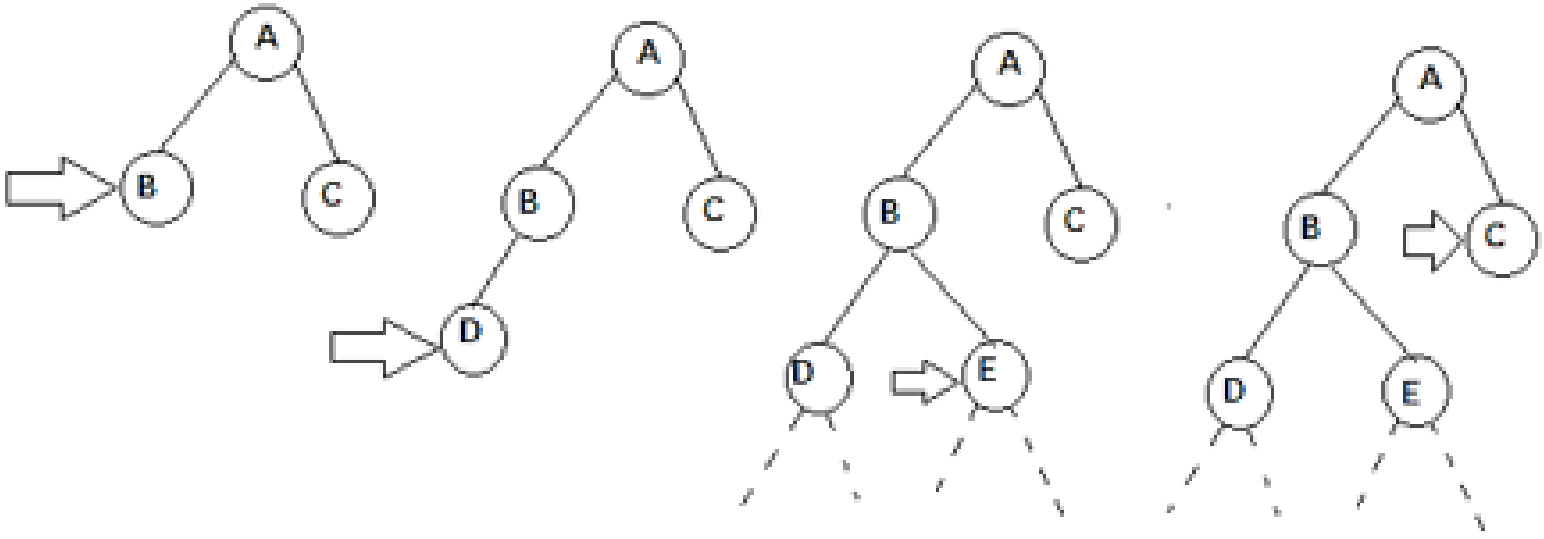
Disadvantages

- This algorithm may not terminate and go on infinitely on one path.
  - The solution to this issue is to **choose a cut-off depth.**
  - If the ideal cut-off is *d*, and if **chosen cut-off is lesser** than *d*, then this algorithm may fail.
  - If **chosen cut-off is more** than *d*, then execution time increases.
- Its complexity depends on the number of paths
- It cannot check duplicate nodes.

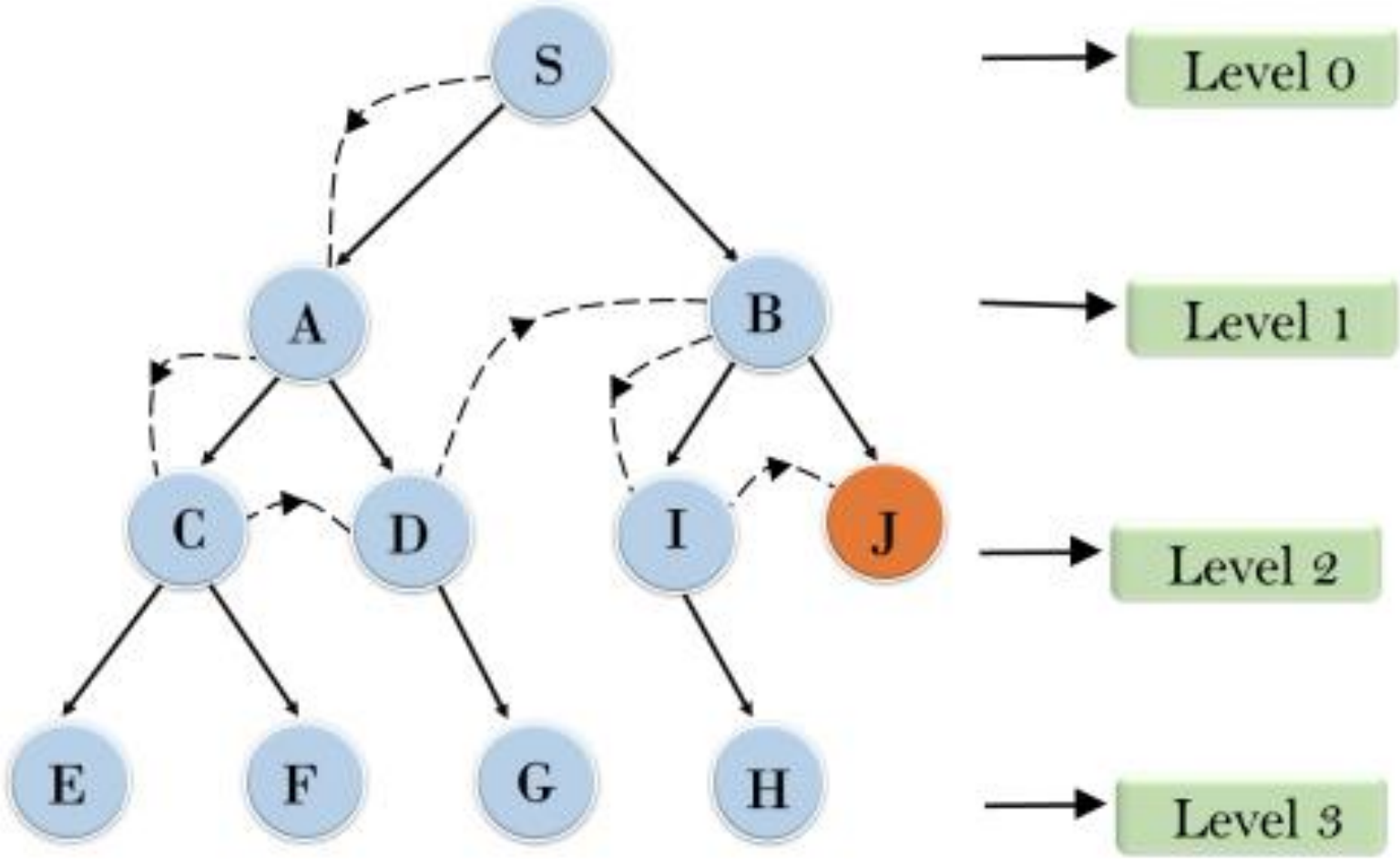
DEPTH LIMITED SEARCH

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a **predetermined depth limit .**
  - Nodes at depth are treated as if they have no successors.
- This approach is called depth-limited search

- Depth-limited search can terminate with two conditions:
  - If the solution is found.
  - If there is no solution within given depth limit.
- **PROCESS**: If depth is fixed to 2, DLS carries out depth first search till second level in the search tree.



Depth Limited Search



Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

- What if solution is deeper than L?
  - Increase L iteratively.

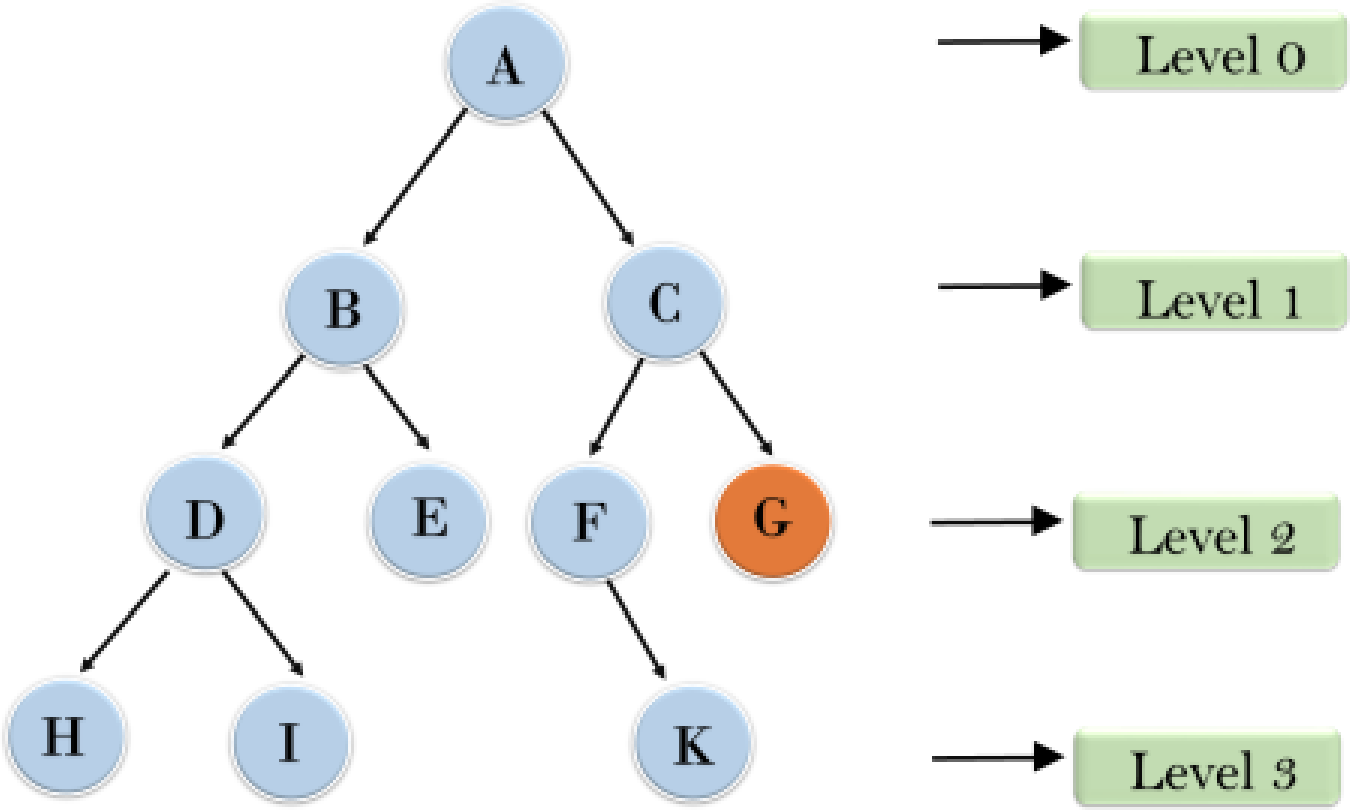
ITERATIVE DEEPENING SEARCH

This inherits the memory advantage of Depth-First search, and is better in terms of time complexity than Breadth first search

ITERATIVE DEEPENING SEARCH

- Iterative deepening depth-first search/ Iterative deepening search [**depth-first tree search that finds the best depth limit**]
  - It does this by gradually increasing the limit—first 0, then 1, then 2,and so on—until a goal is found.
- This will occur when the depth limit reaches d, the depth of the shallowest goal node.
- Iterative deepening combines the benefits of depth-first and breadth-first search.

Iterative deepening depth first search



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G

Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

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.

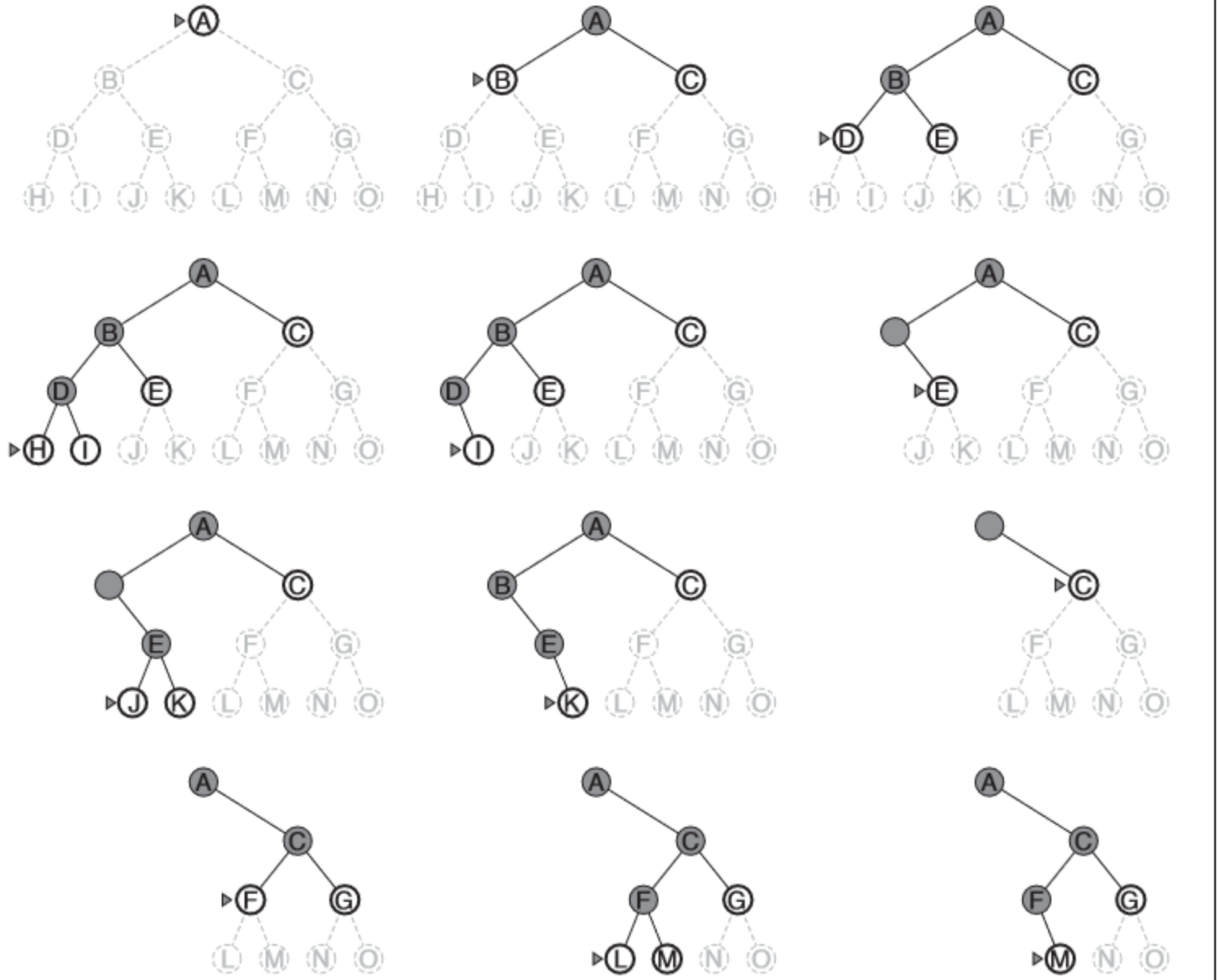
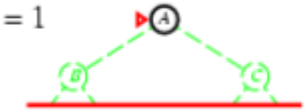


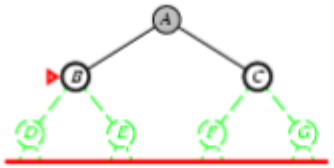
Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

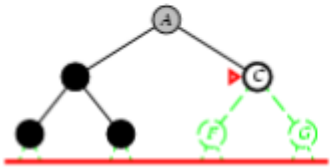
Limit = 0

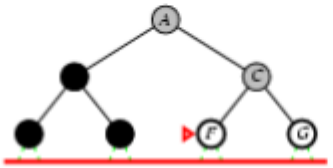
Limit = 1

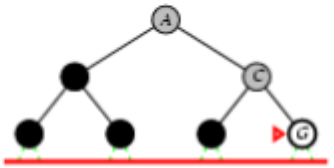


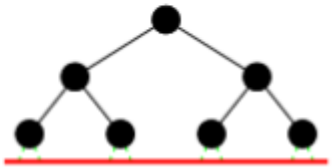
Limit = 2



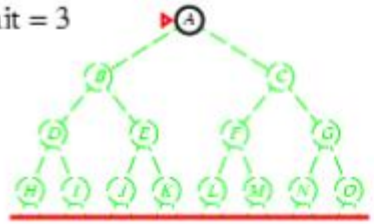


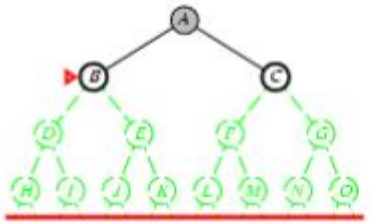


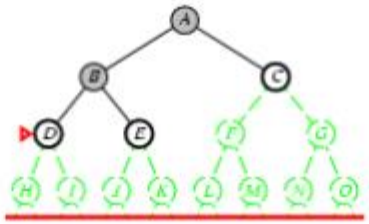


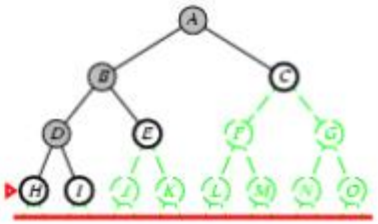


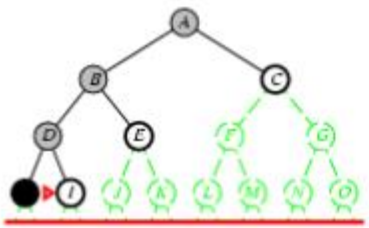
Limit = 3

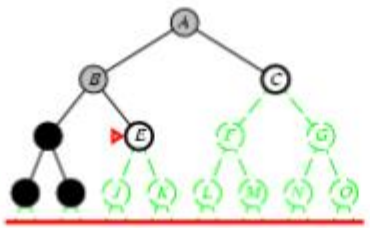


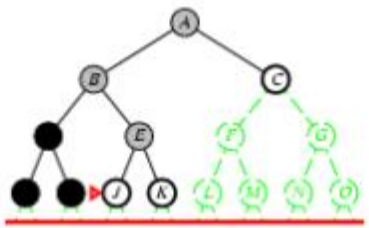


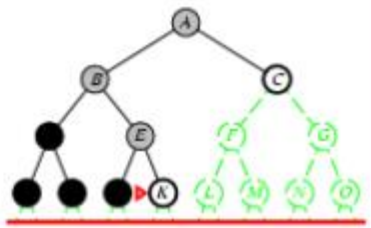


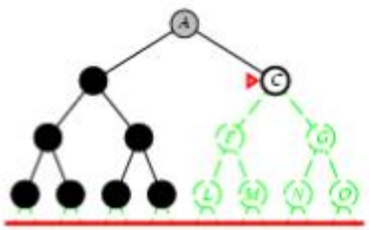


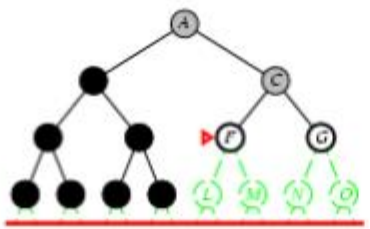


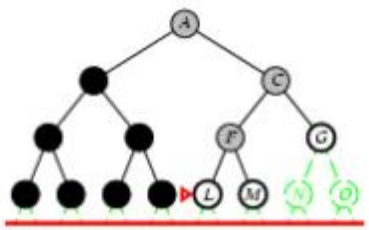


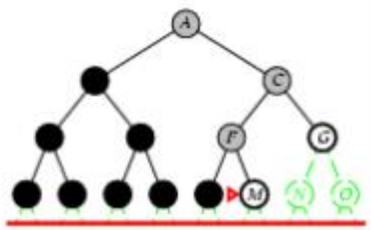












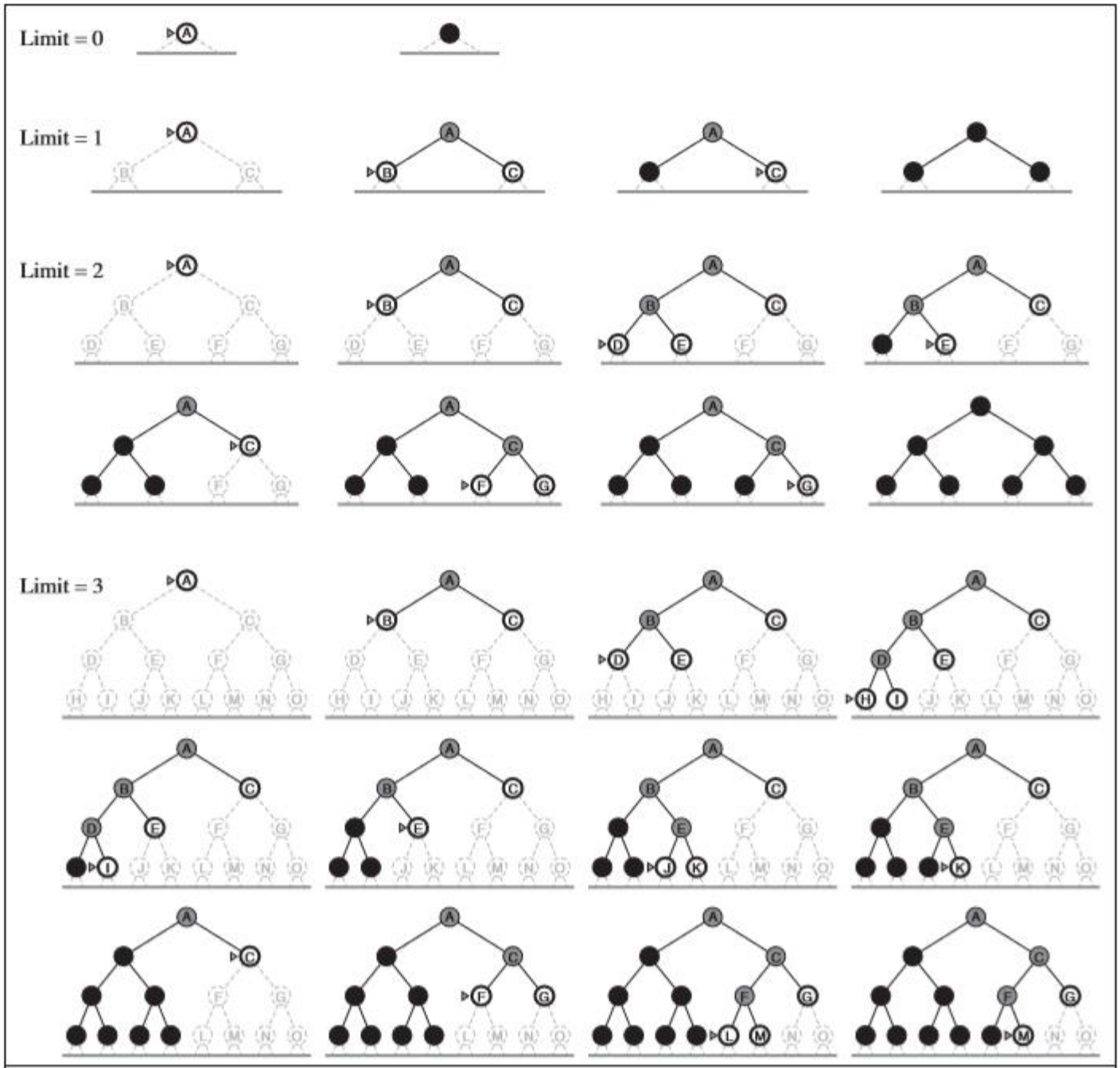


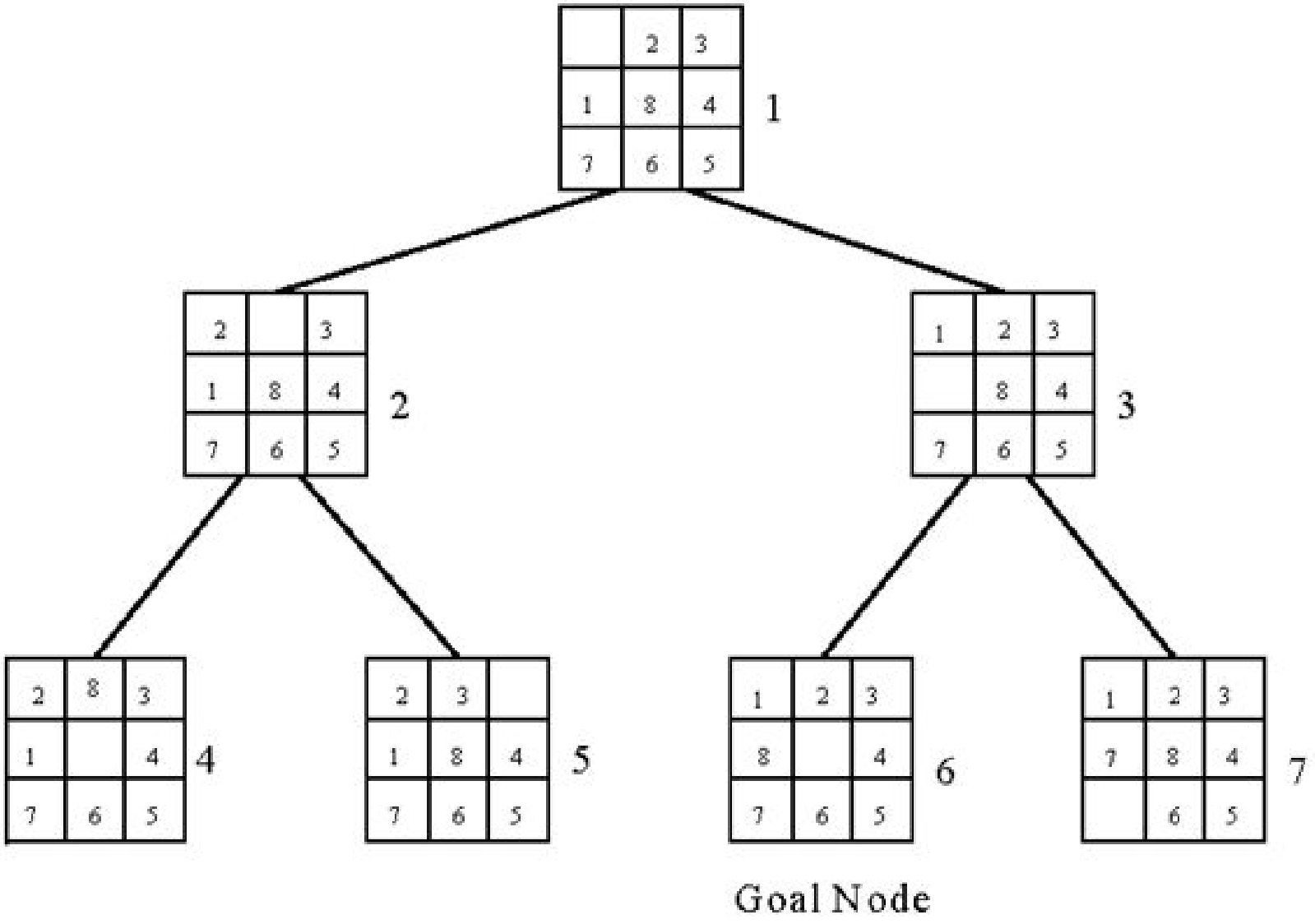
Figure 3.19 Four iterations of iterative deepening search on a binary tree.

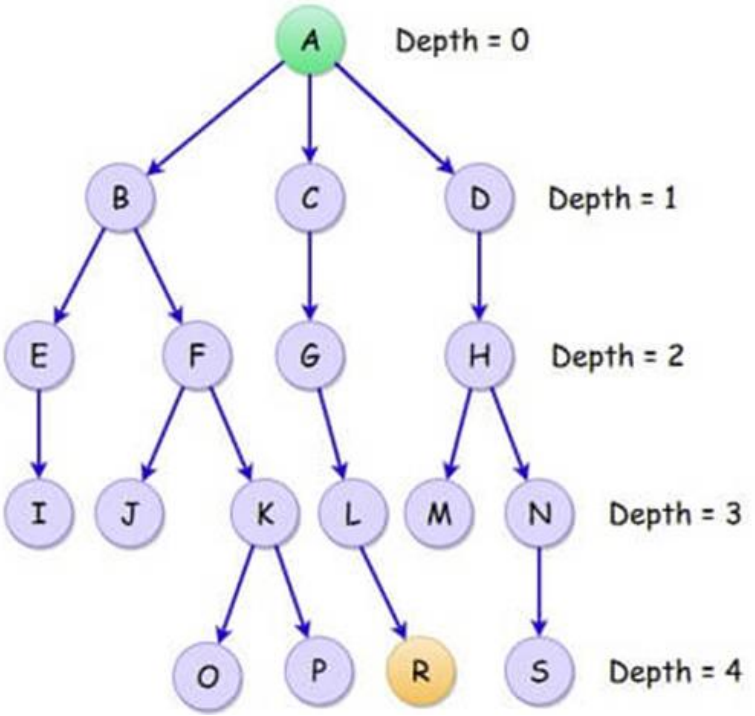
BFS,DFS

	2	3
1	8	4
7	6	5

1	2	3
8		4
7	6	5

8-puzzle problem





Solve for IDDS with all steps

The tree can be visited as: A B E F C G D H

DEPTH = {0, 1, 2, 3, 4}

DEPTH LIMITS	IDDFS
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G L D H M N
4	A B E I F J K O P C G L R D H M N S

BFS,DFS,DLS,IDS

1	2	3
	4	6
7	5	8

1	2	3
4	5	6
7	8	

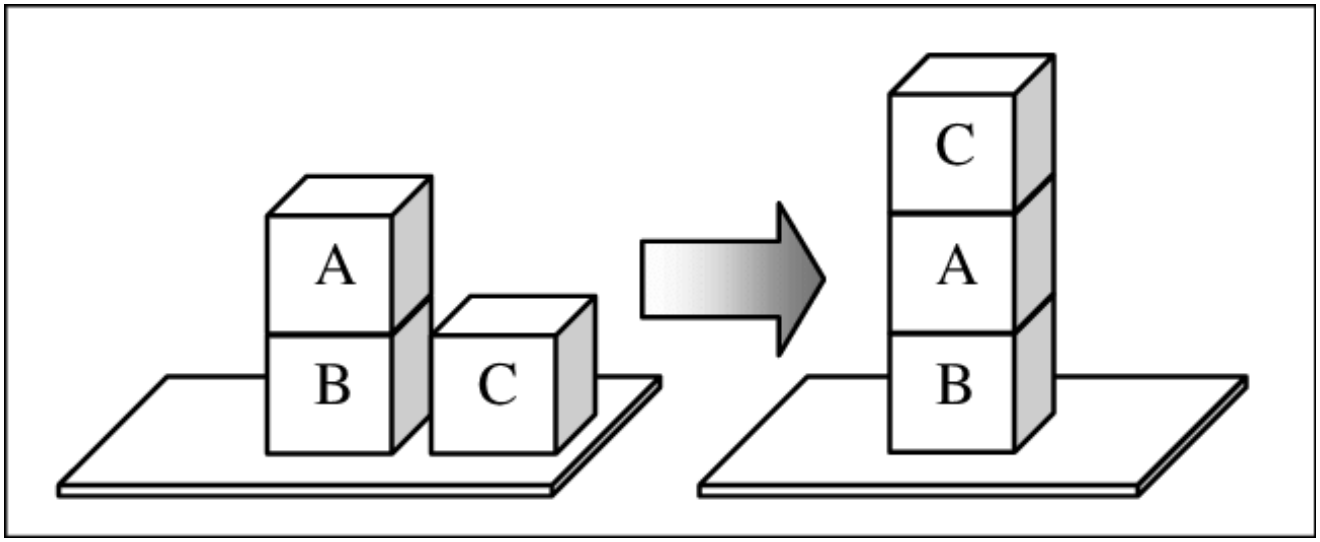
BFS Sequence-

DFS Sequence-

DLS Sequence-

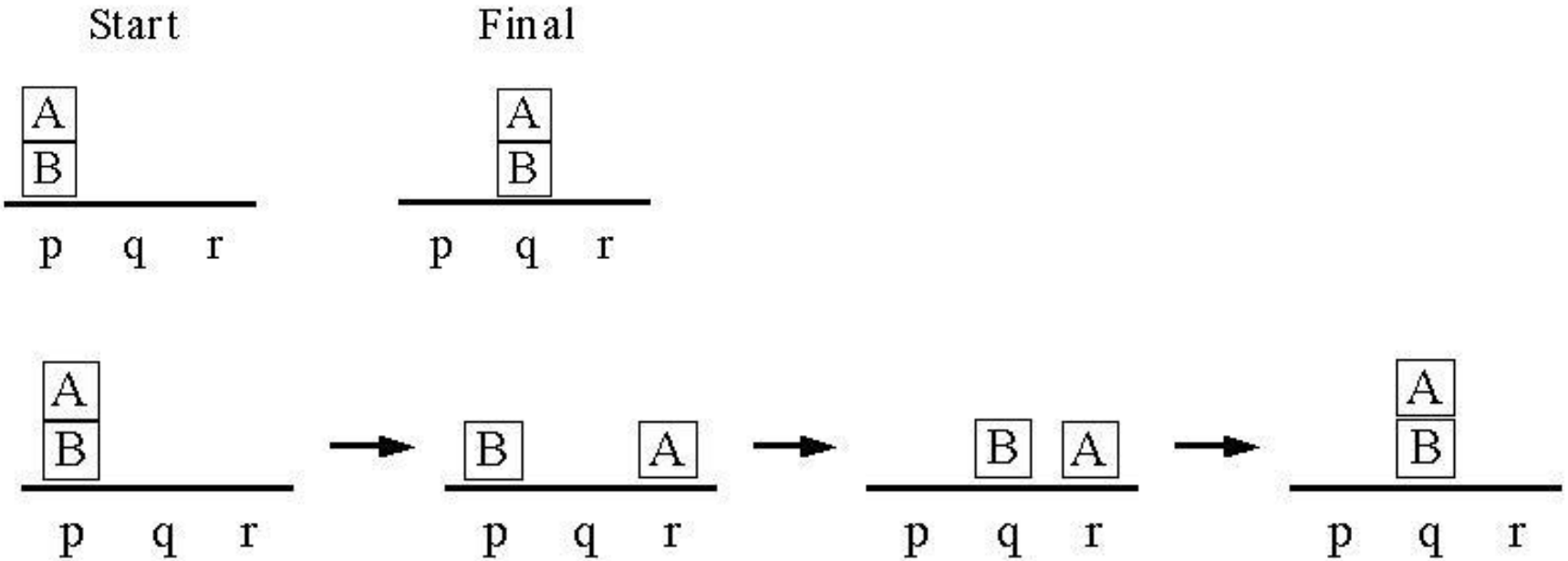
IDS Sequence-

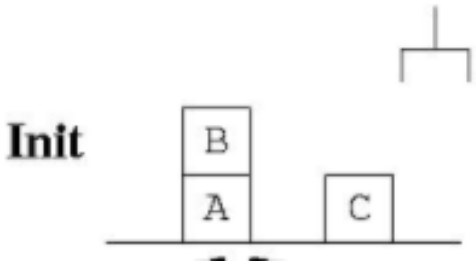
Example- BLOCKS WORLD

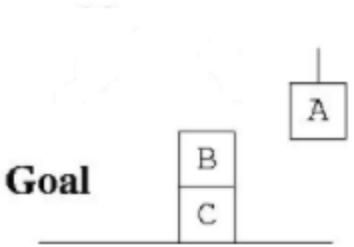


The world consists of:

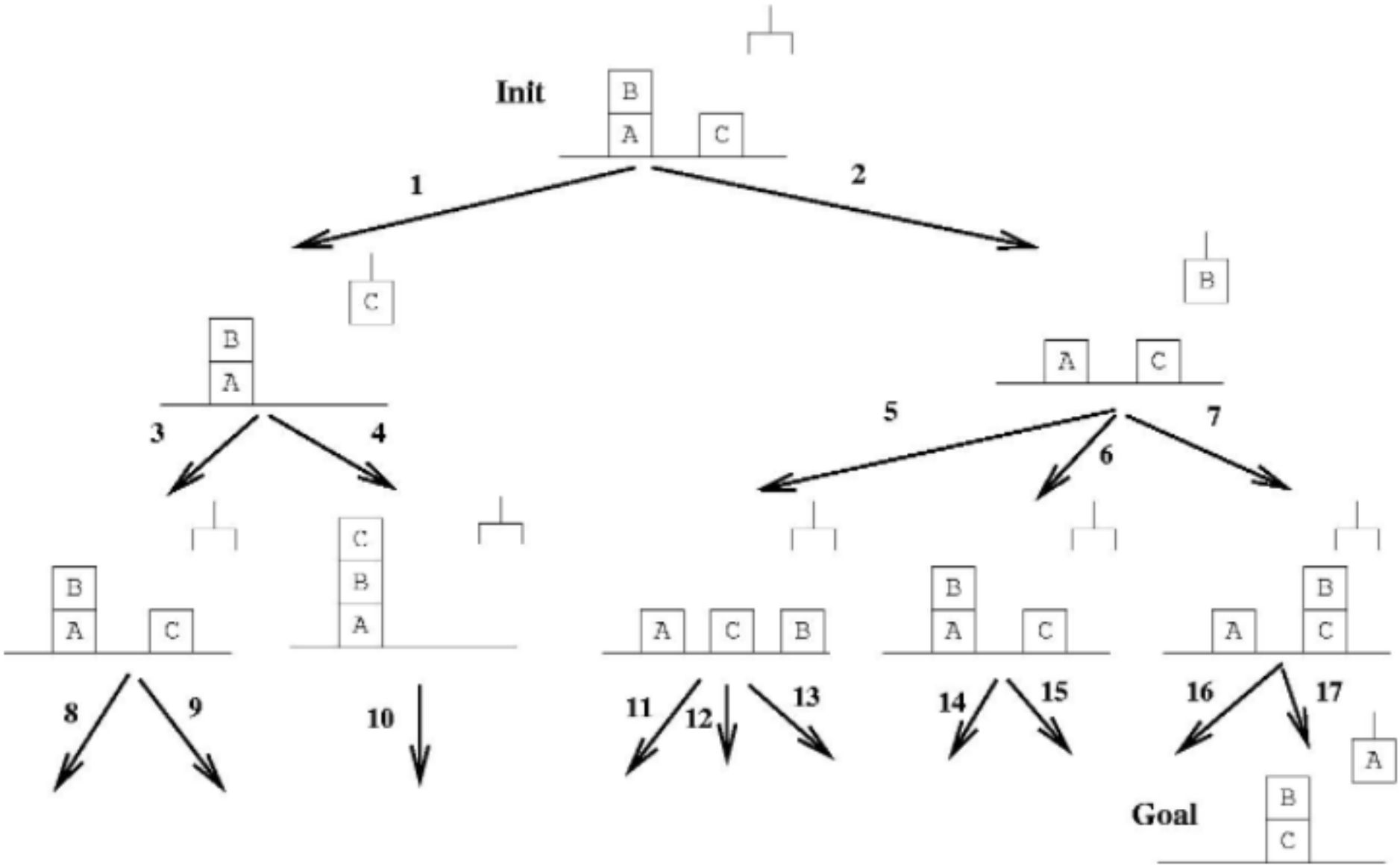
- A flat surface such as a table top
- An adequate set of identical blocks which are identified by letters.
- The blocks can be stacked one on one to form towers of apparently unlimited height.


Start
Final
A
B
p q r
A
B
p q r
A
B
p q r
B A
p q r
B A
p q r
A
B
p q r

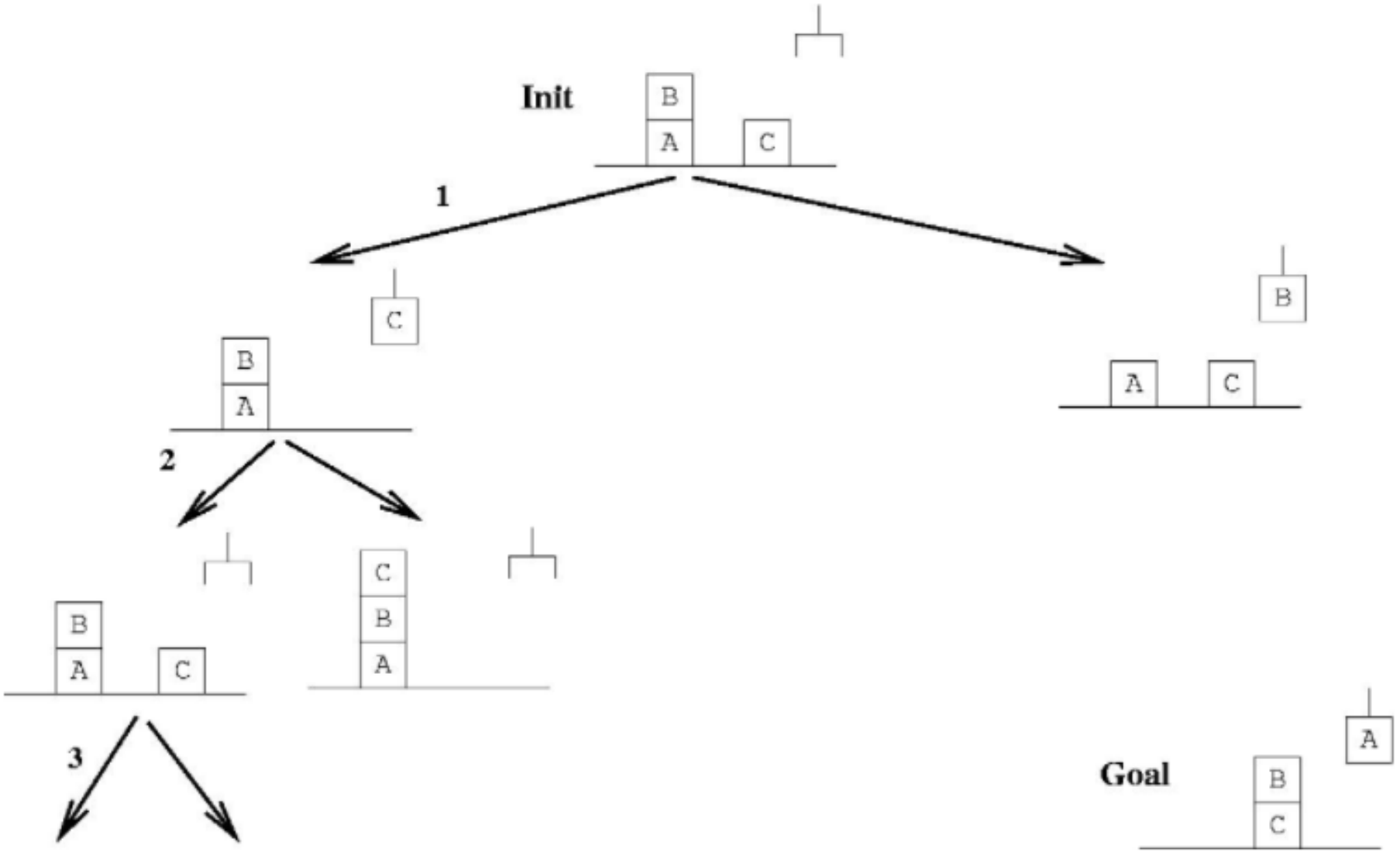

Init
B
A
C


Goal
A
B
C

An Example for Breadth-first search



An Example for Depth-first search



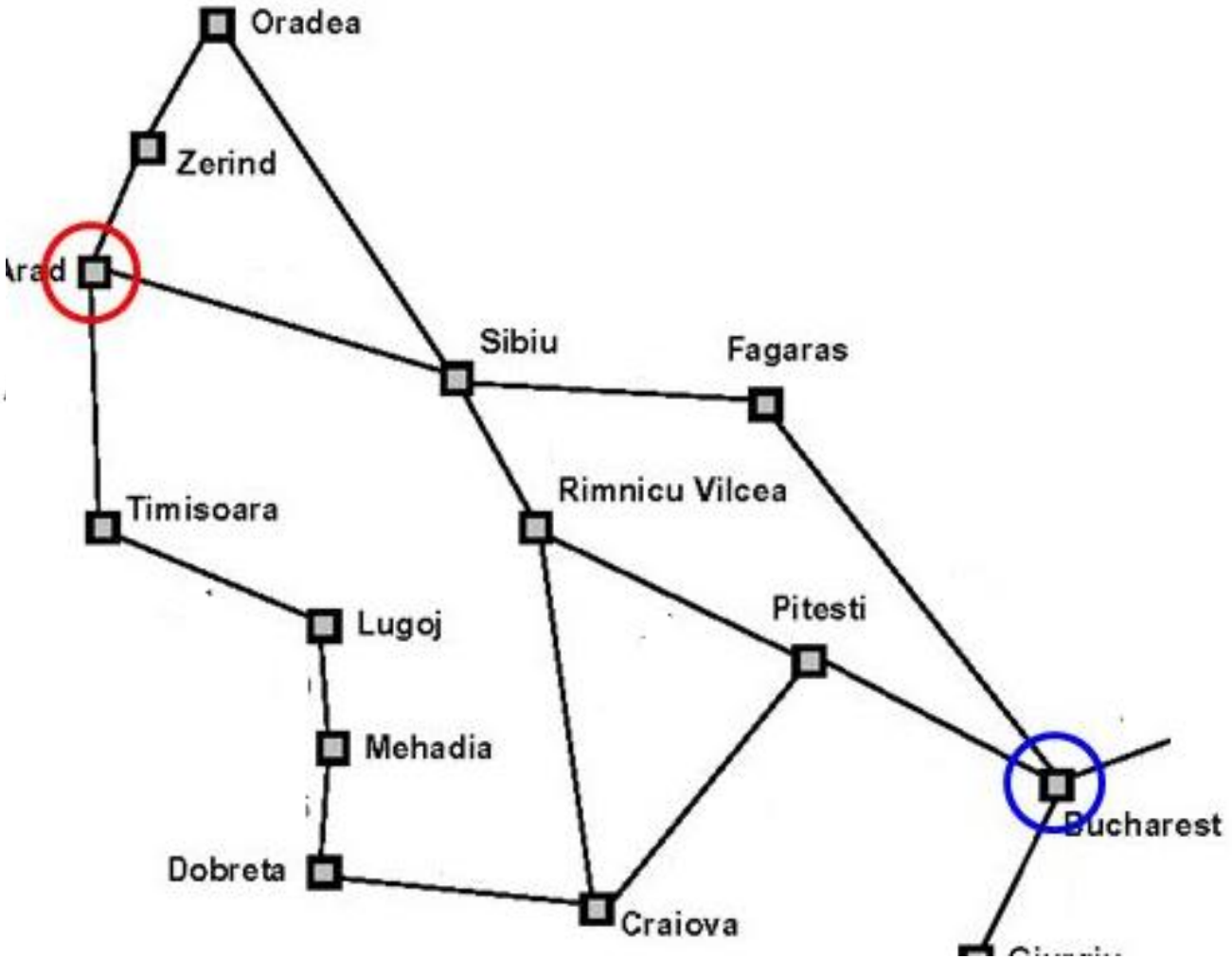
Romania Problem

Initial state: Arad

Goal state: Bucharest

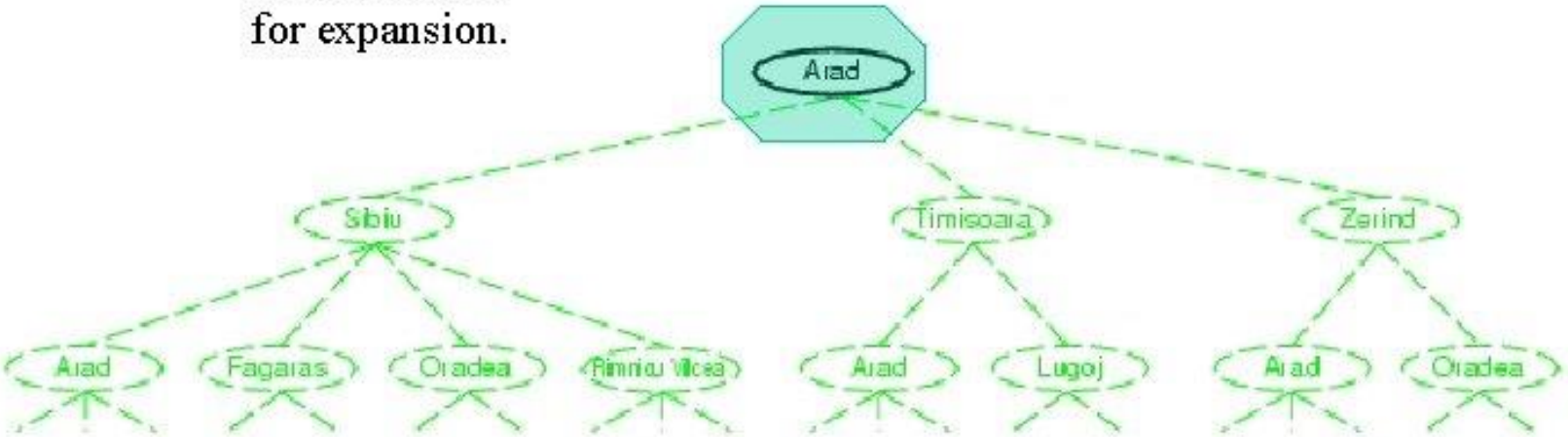
Operators: From any node, you can visit any connected node.

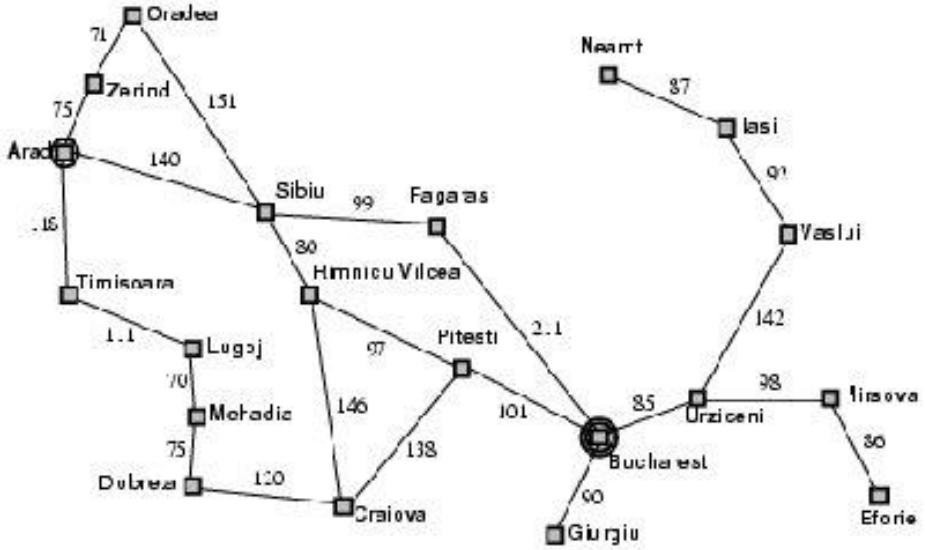
Operator cost, the driving distnace.



Tree search example

Node selected for expansion.



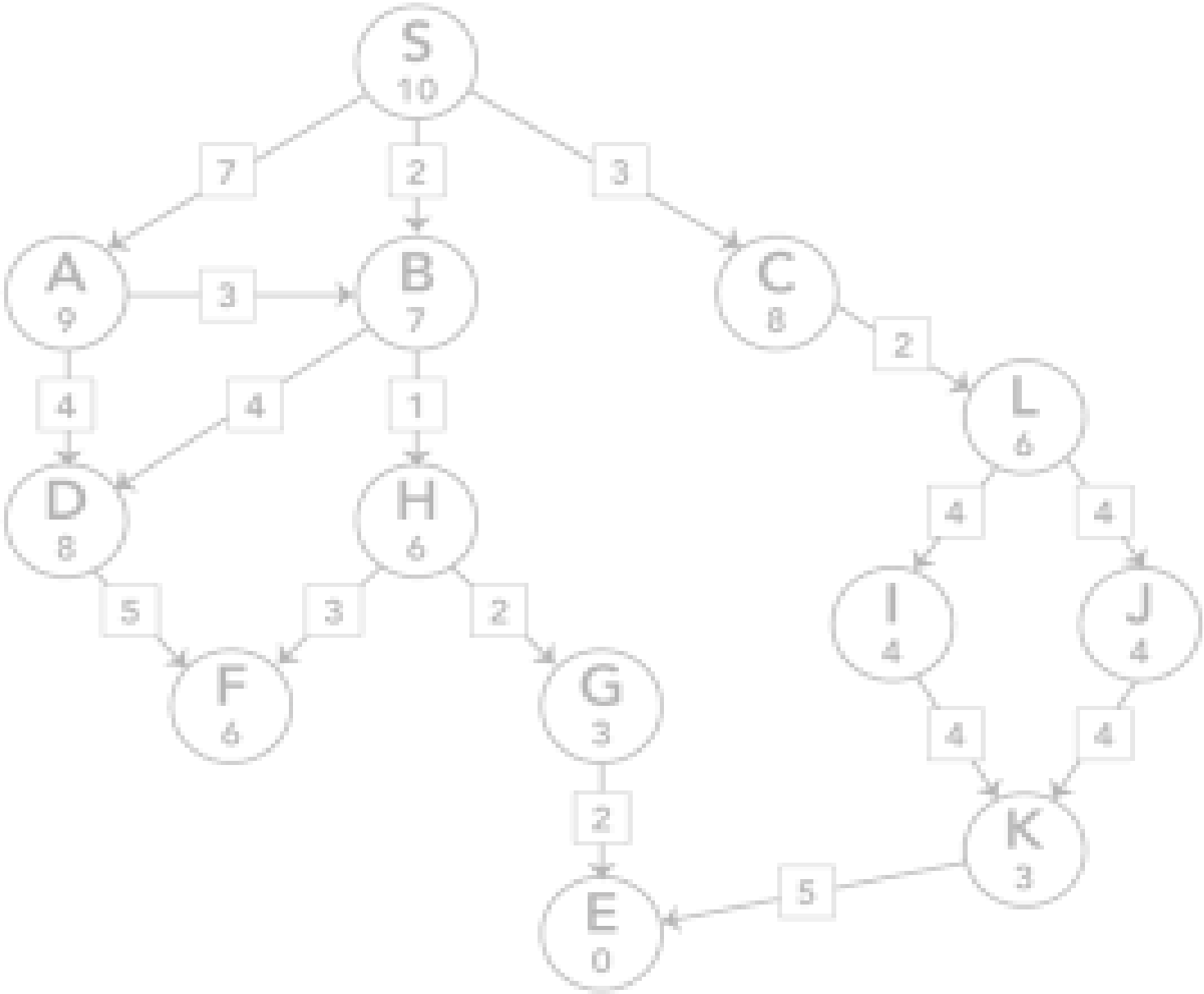


BFS Sequence-

DFS Sequence-

DLS Sequence-

IDS Sequence-


S
10
7
2
3
A
9
3
B
7
C
8
4
4
1
2
L
6
D
8
H
6
4
4
5
3
2
I
4
J
4
F
4
G
3
4
4
2
K
3
5
E
0

INFORMED SEARCH

- Informed search strategy
  - one that uses problem-specific knowledge beyond the definition of the problem itself
  - Find solutions more efficiently than an uninformed strategy.

BEST-FIRST SEARCH

- **Idea**: Use an evaluation function f(n) for each node
  - f(n) provides an estimate for the **total cost.**

Expand the node n with smallest f(n).

Implementation:

Order the nodes in increasing order of cost.

SPECIAL CASES

- Greedy best-first search
- A* search ,AO*

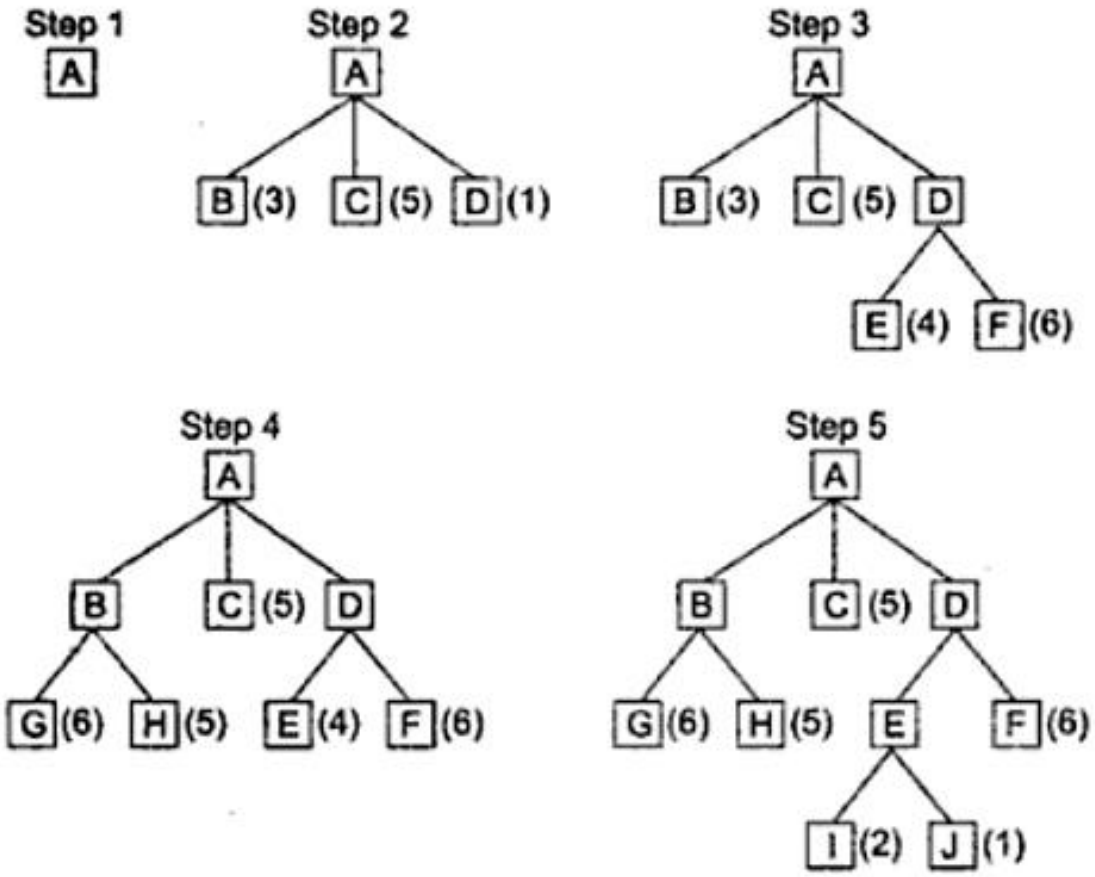
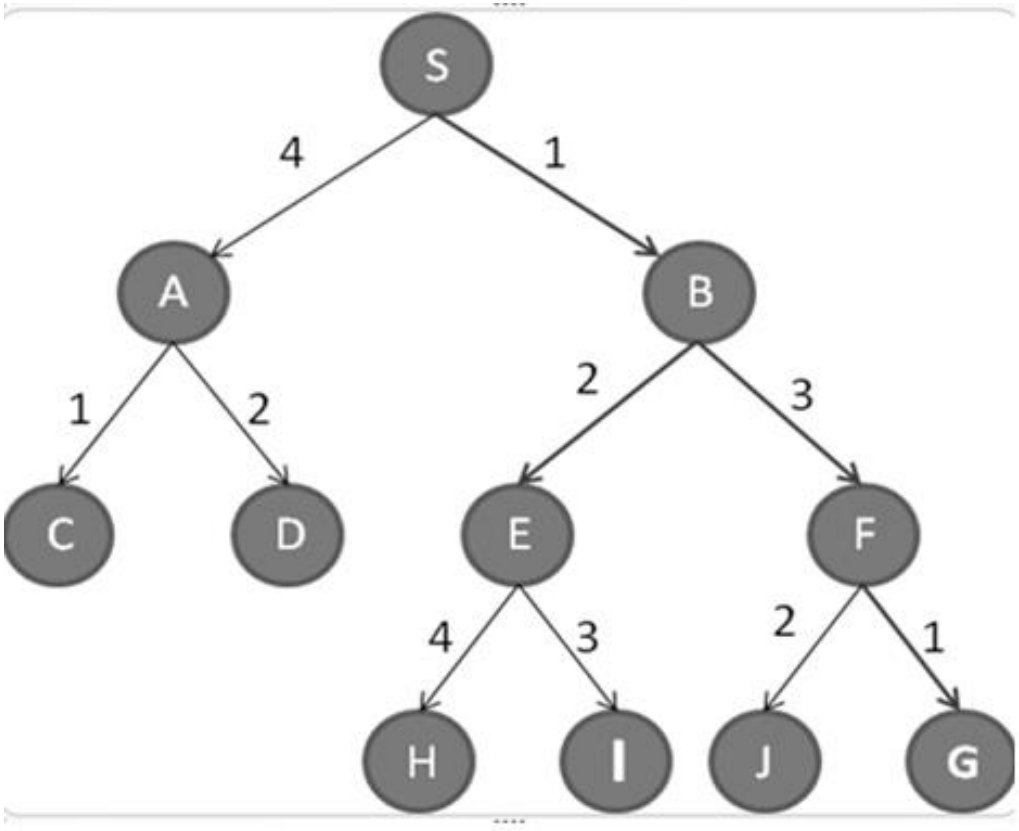


Fig. 3.3 *A Best-First Search*



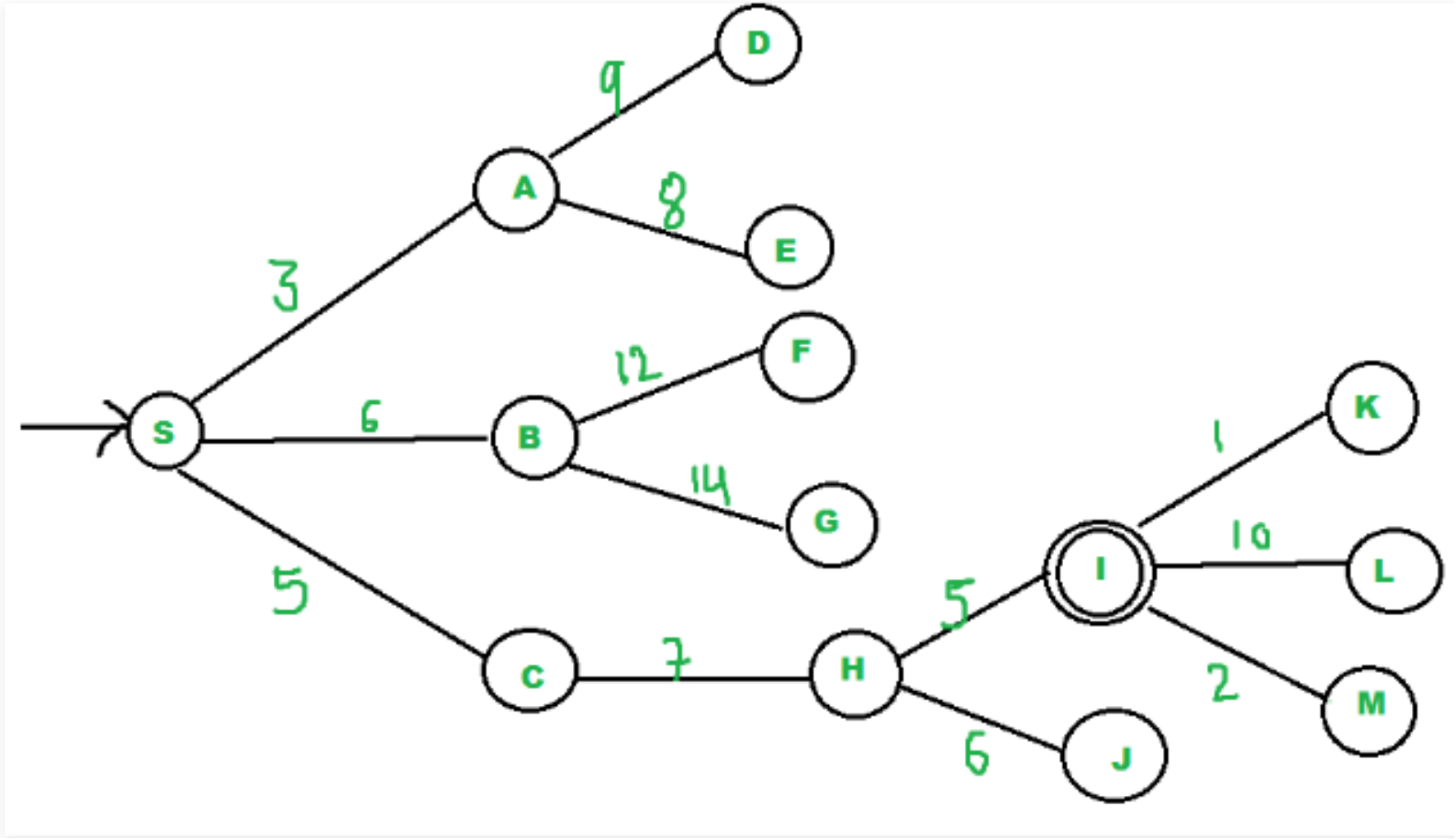
Source : S

Goal: G

The shortest path form S -> G is S -> B -> F -> G

Total cost: 1+3+1=5

EXAMPLE



GREEDY BEST-FIRST SEARCH

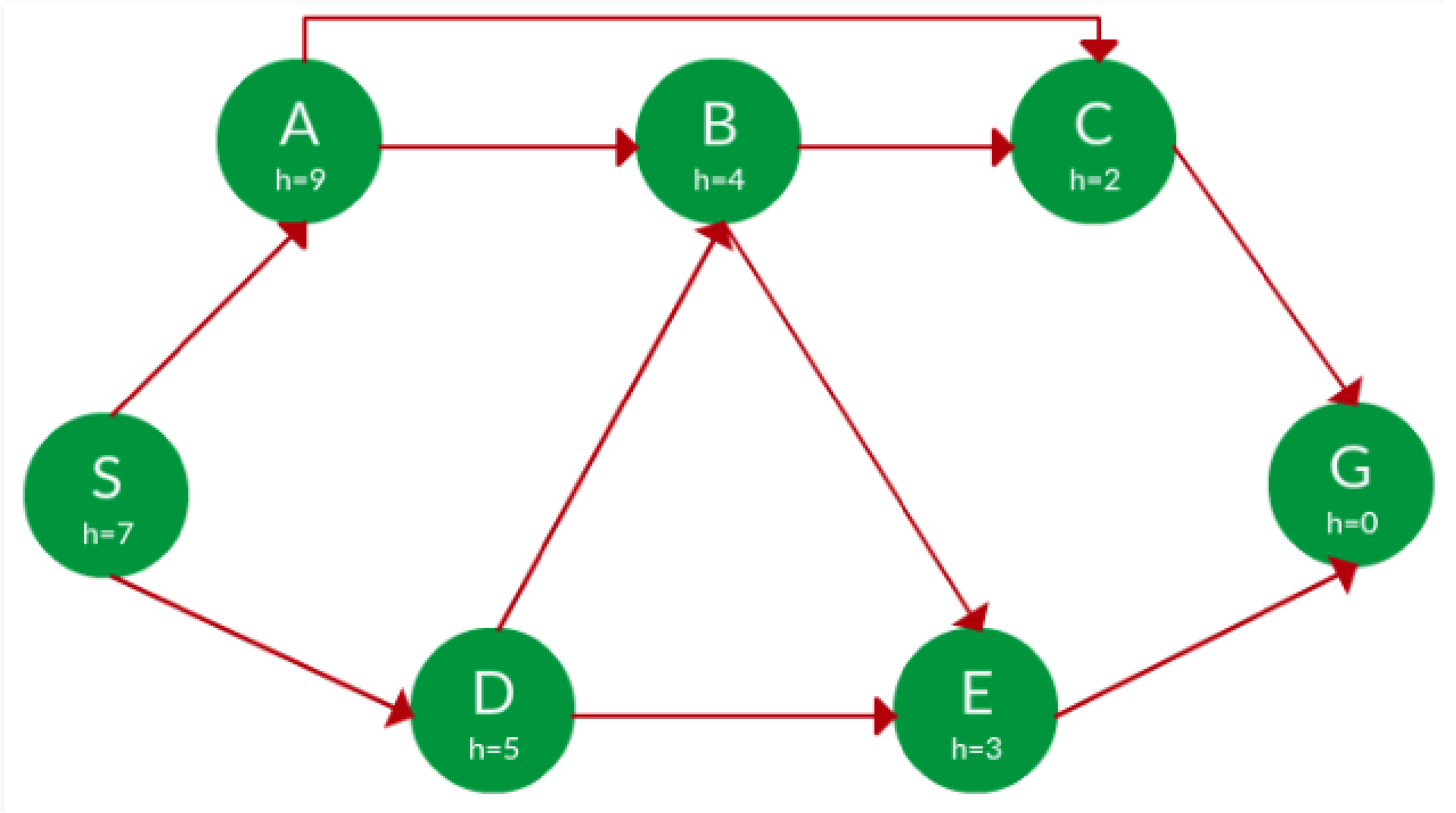
- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- Evaluates nodes by using **heuristic function**; that is**, f(n)=h(n).**
- Algorithm is called “greedy”—at each step it tries to get as close to the goal as it can.

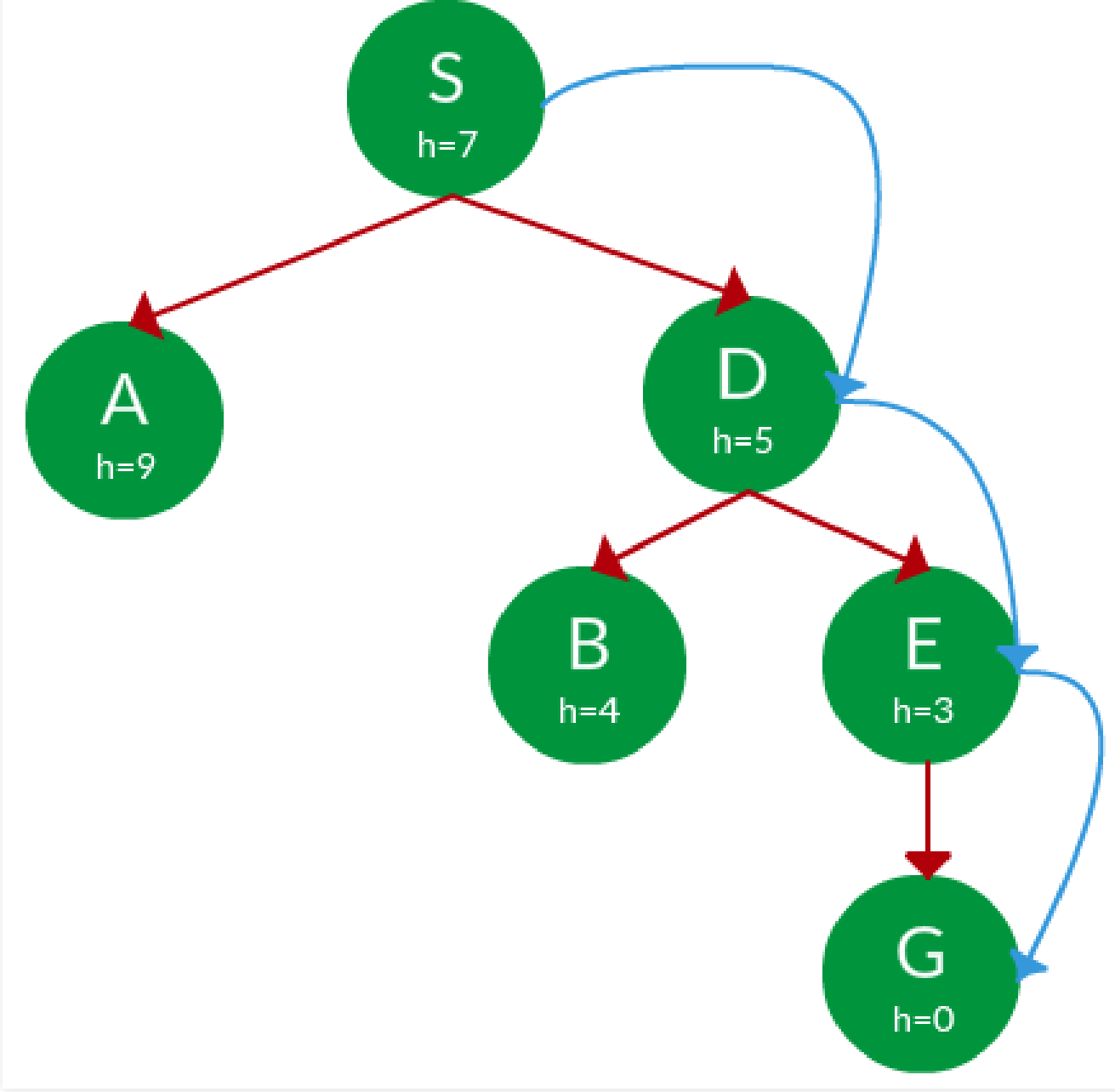
GREEDY BEST FIRST SEARCH

- Expand the node closest to the goal node.
  - The “closeness” is estimated by a heuristic h(x) .
  - **Heuristic function** is a way to inform the search about the direction to a goal.
  - It provides an informed way to guess which neighbour of a node will lead to a goal.
- **Heuristic:** A heuristic h is defined as-

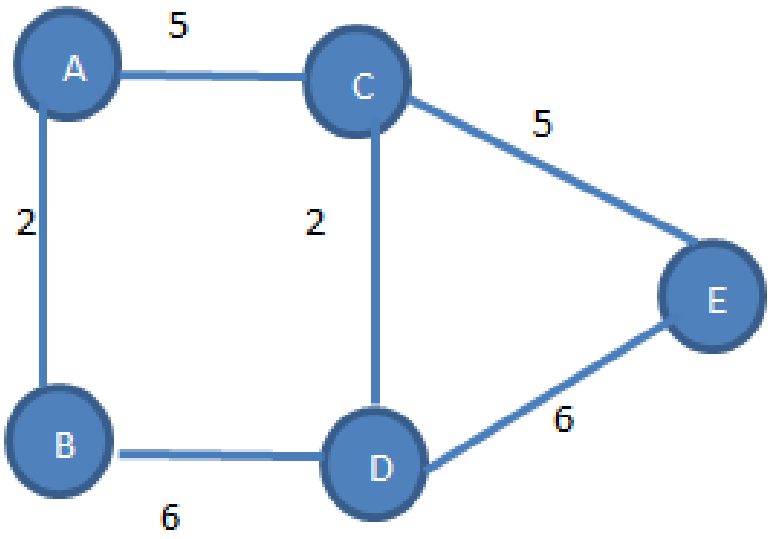
  h(x) = Estimate of distance of node x from the goal node.
  - Lower the value of h(x), closer is the node from the goal.

  Strategy: Expand the node closest to the goal state, *i.e.* expand the node with lower h value.


A
h=9
B
h=4
C
h=2
S
h=7
G
h=0
D
h=5
E
h=3

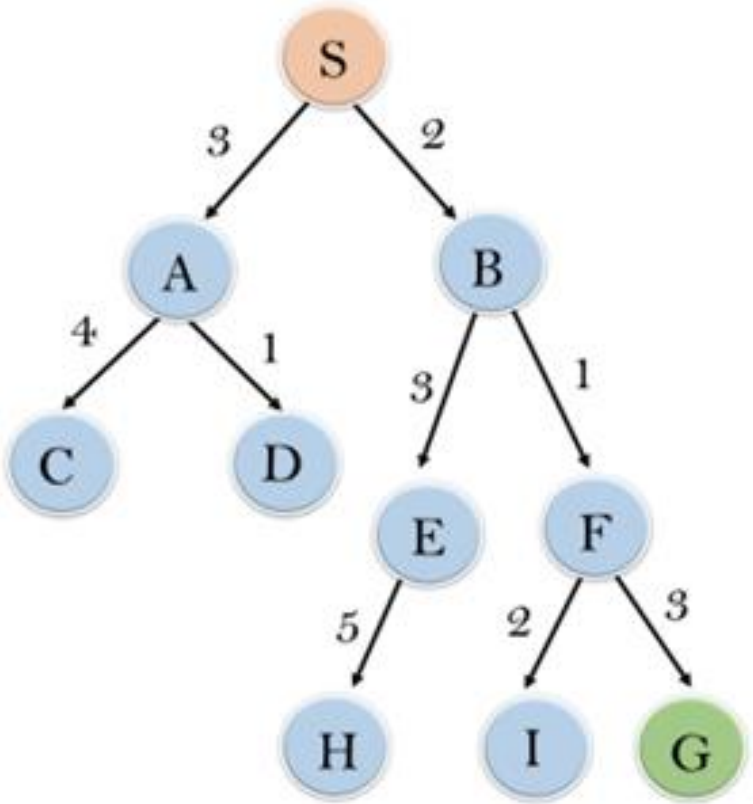


Path: `S -> D -> E -> G`



Node	Heuristic value
A	10
B	8
D	3
C	5
E	0

A --- C ---- E



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

S----> B----->F----> G

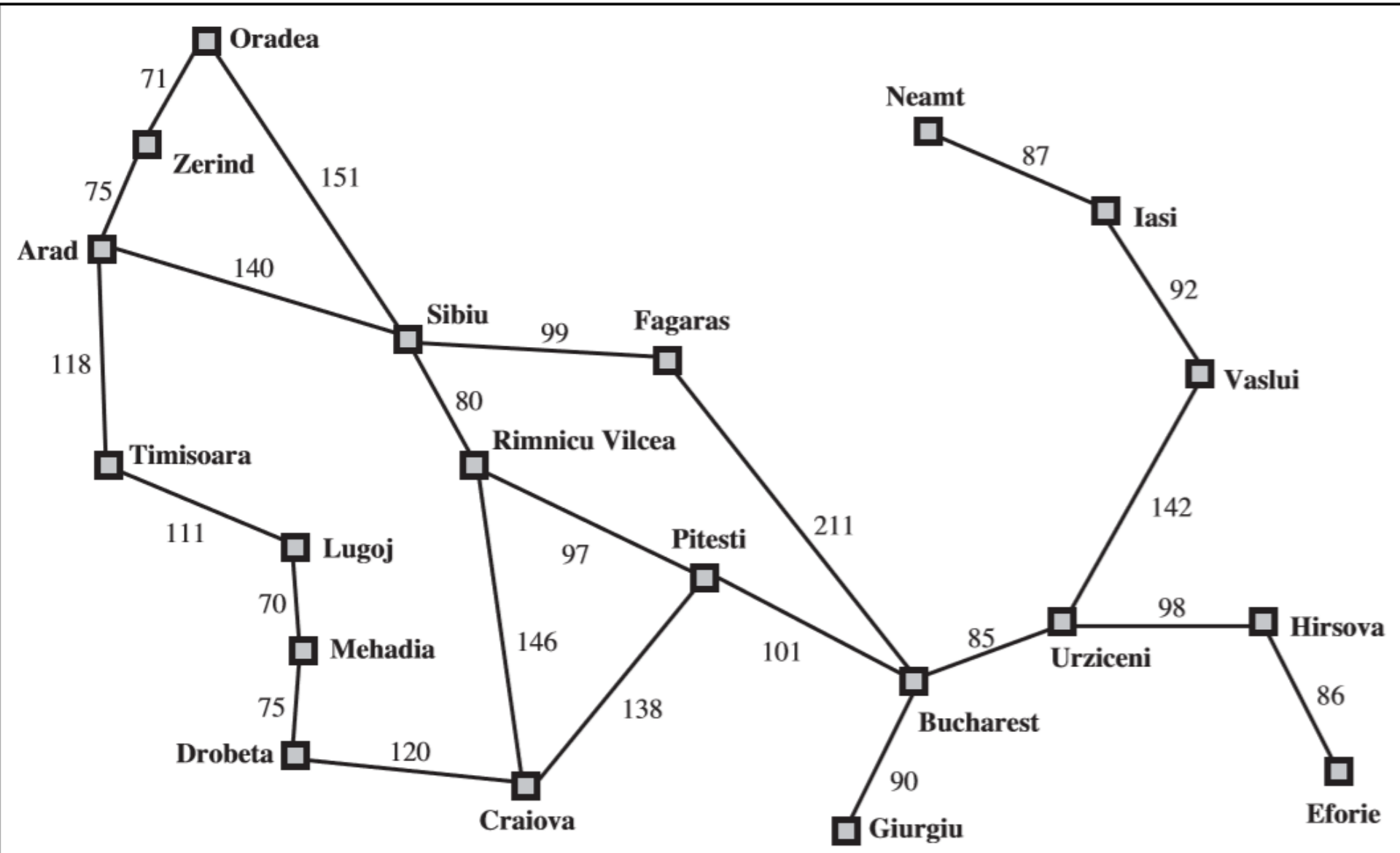
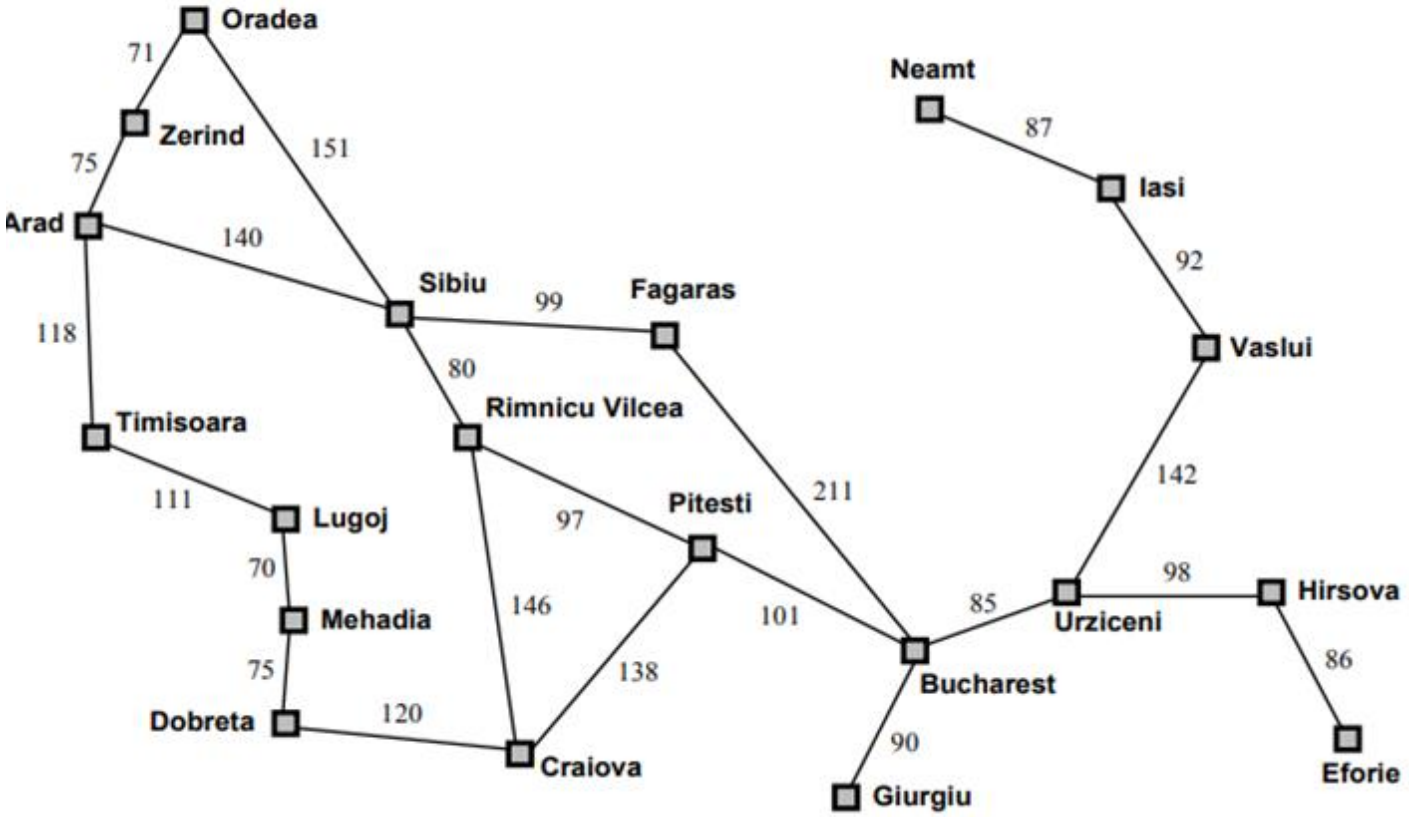


Figure 3.2 A simplified road map of part of Romania.



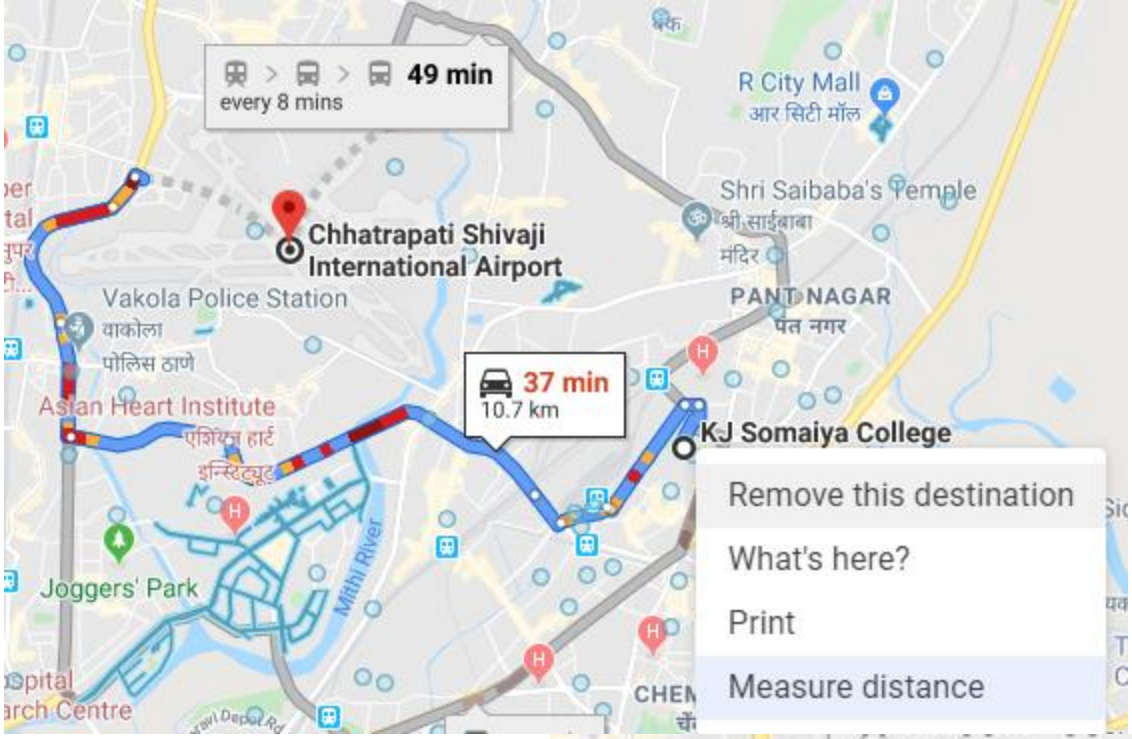
Solve for Best First Search technique

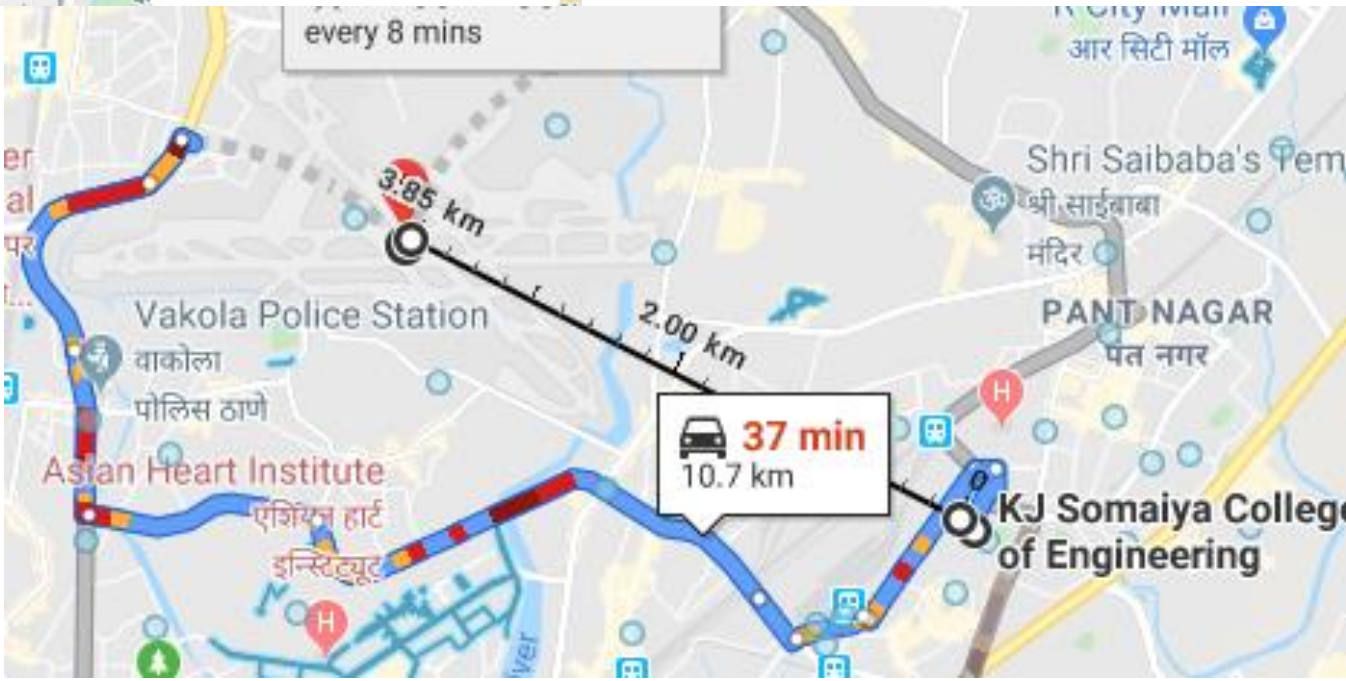
A– Z – O – S – R – P – B = 575

Straight-line Distance as Heuristic

Straight line distances as an admissible **heuristic** as they will never overestimate the cost to the goal.

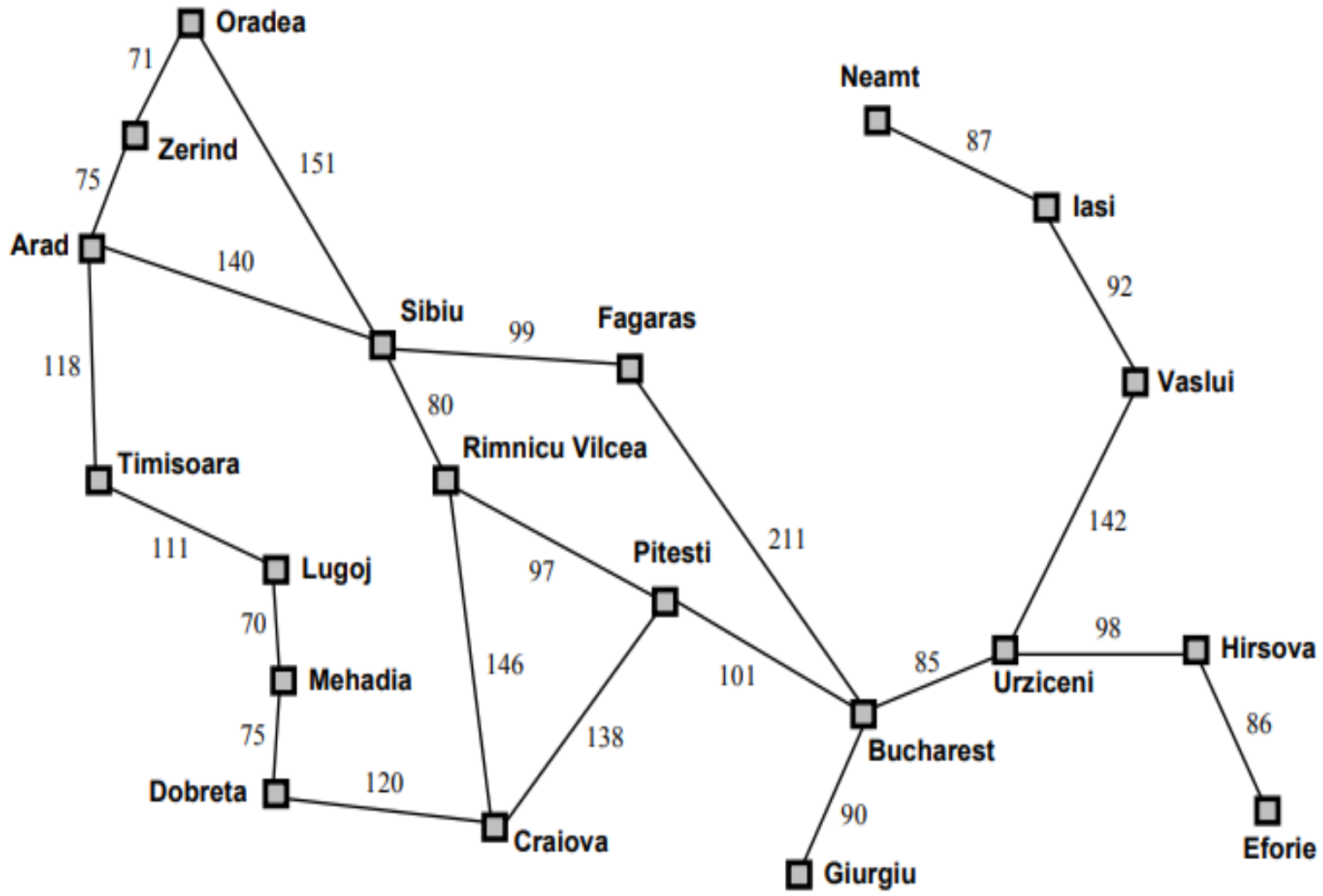
This is because there is no shorter **distance** between two cities than the **straight line distance**.


49 min
every 8 mins
R City Mall
आर सिटी मॉल
Shri Saibaba's Temple
श्री साईबाबा मंदिर
Chhatrapati Shivaji International Airport
Vakola Police Station
वाकोला पोलिस ठाणे
PANT NAGAR
पंत नगर
37 min
10.7 km
Asian Heart Institute
एशियन हार्ट इन्स्टिट्यूट
KJ Somaiya College
Remove this destination
What's here?
Print
Measure distance
Joggers' Park
Mithi River


every 8 mins
आर सिटी मॉल
Shri Saibaba's Tem
श्री साईबाबा मंदिर
3.85 km
2.00 km
Vakola Police Station
वाकोला पोलिस ठाणे
PANT NAGAR
पंत नगर
37 min
10.7 km
Asian Heart Institute
एशियन हार्ट इन्स्टिट्यूट
KJ Somaiya College of Engineering

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

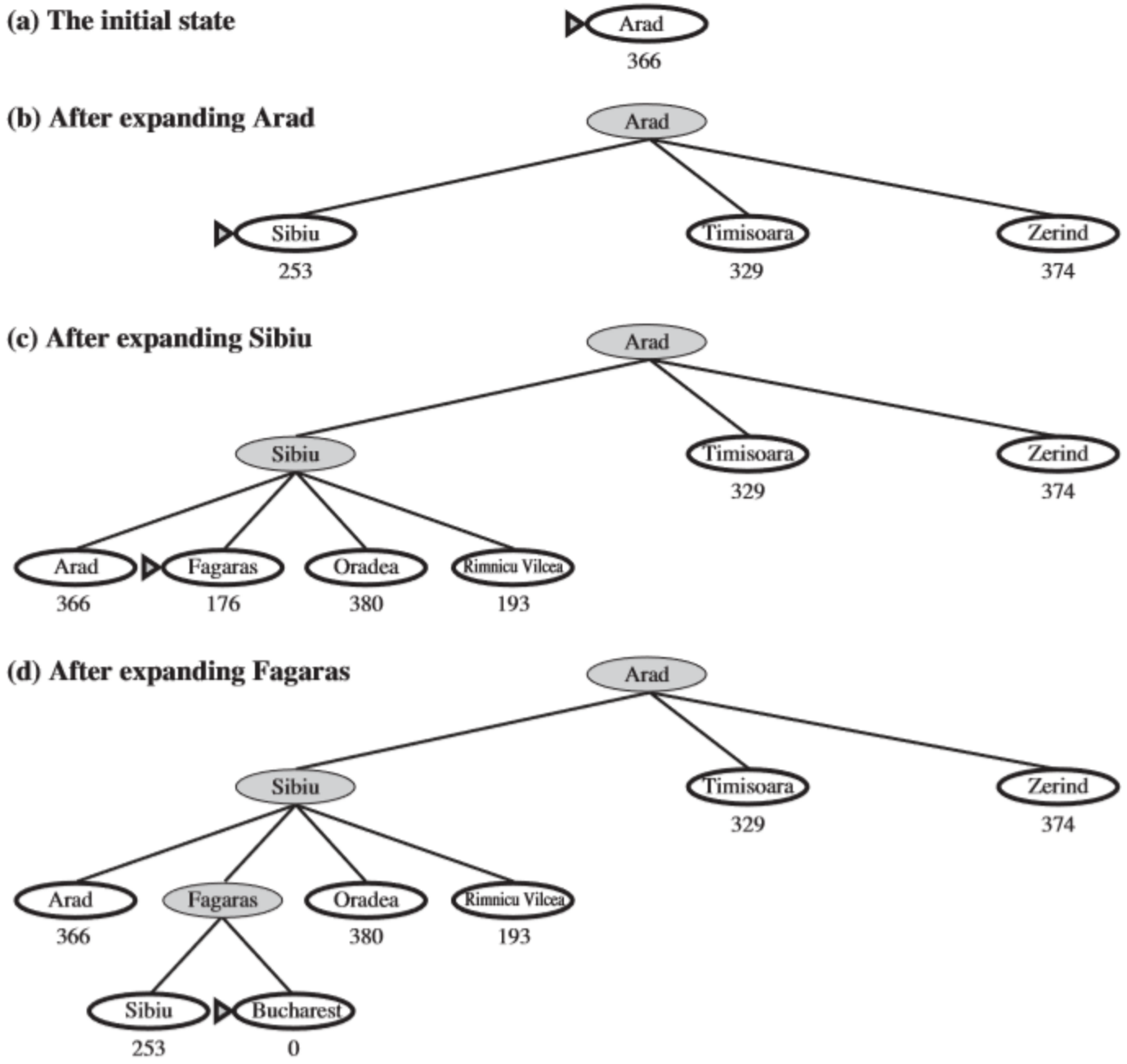
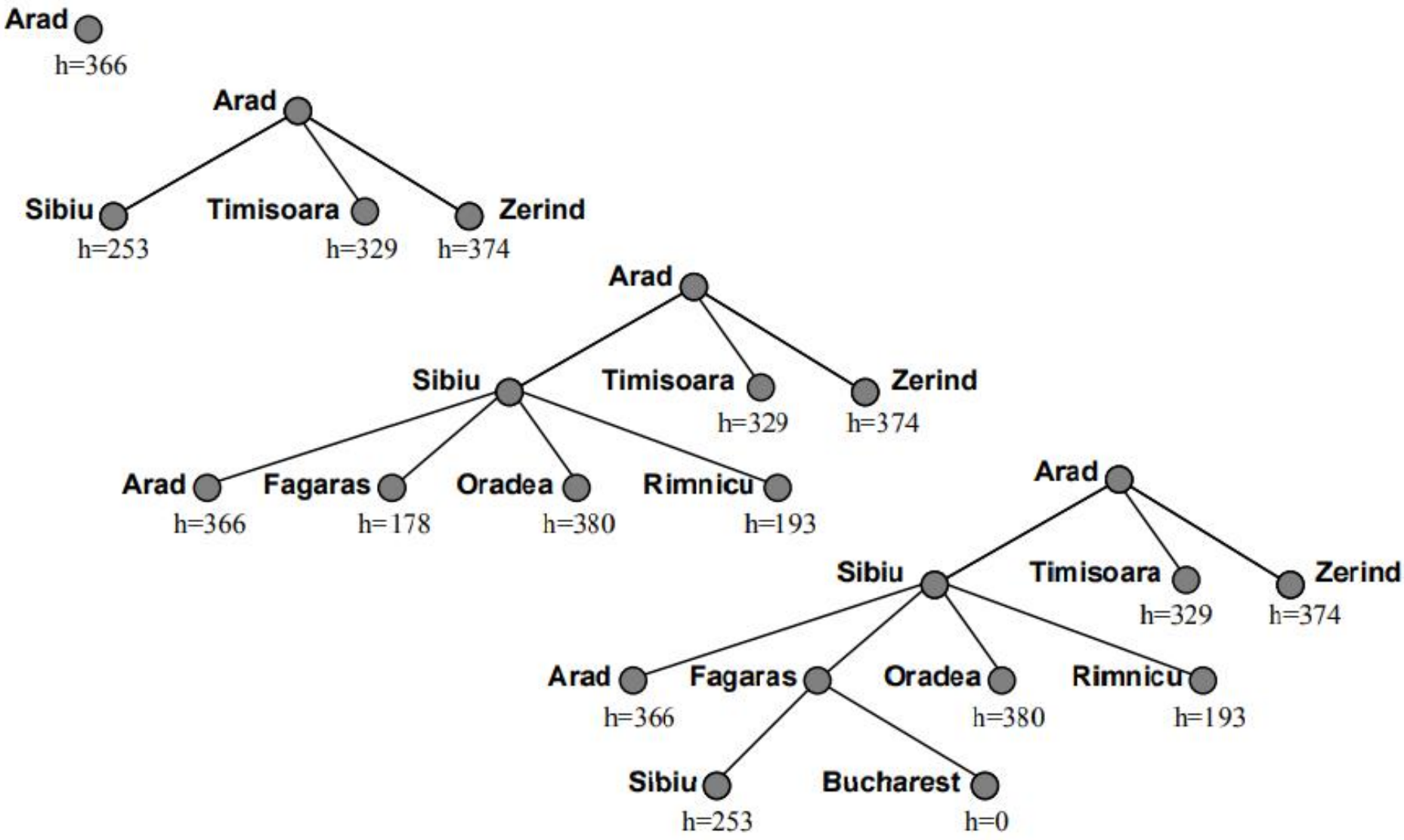


Figure 3.23 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.


Arad
h=366
Arad
Sibiu
Timisoara
Zerind
h=253
h=329
h=374
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=178
h=380
h=193
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=380
h=193
Sibiu
Bucharest
h=253
h=0

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

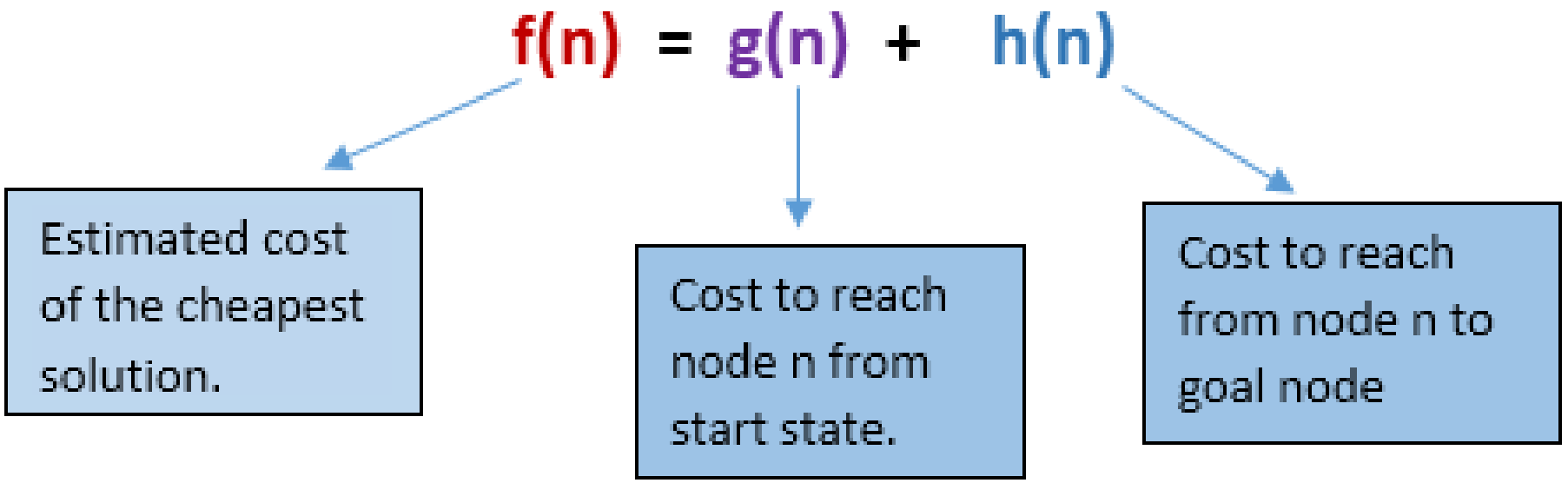
Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal

Arad→Sibiu → Rimnicu Virea → Pitesti → Bucharest is shorter!

A * SEARCH

- AIM: - Minimizing the total estimated solution cost
  - At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.
  - Most widely known form of best-first search is called A∗ search



- F – **F is the** parameter of A* which is the sum of the other parameters G and H and is the **least cost from one node to the next node**.
  - This parameter is responsible for helping us find the most optimal path from our source to destination.
- G – **G is the cost of moving from one node to the other node**.
  - This parameter changes for every node as we move up to find the most optimal path.
- H – **H is the heuristic/estimated path between the current code to the destination node**.
  - This cost is not actual but is, in reality, a guess cost that we use to find which could be the most optimal path between our source and destination.

Step-01:

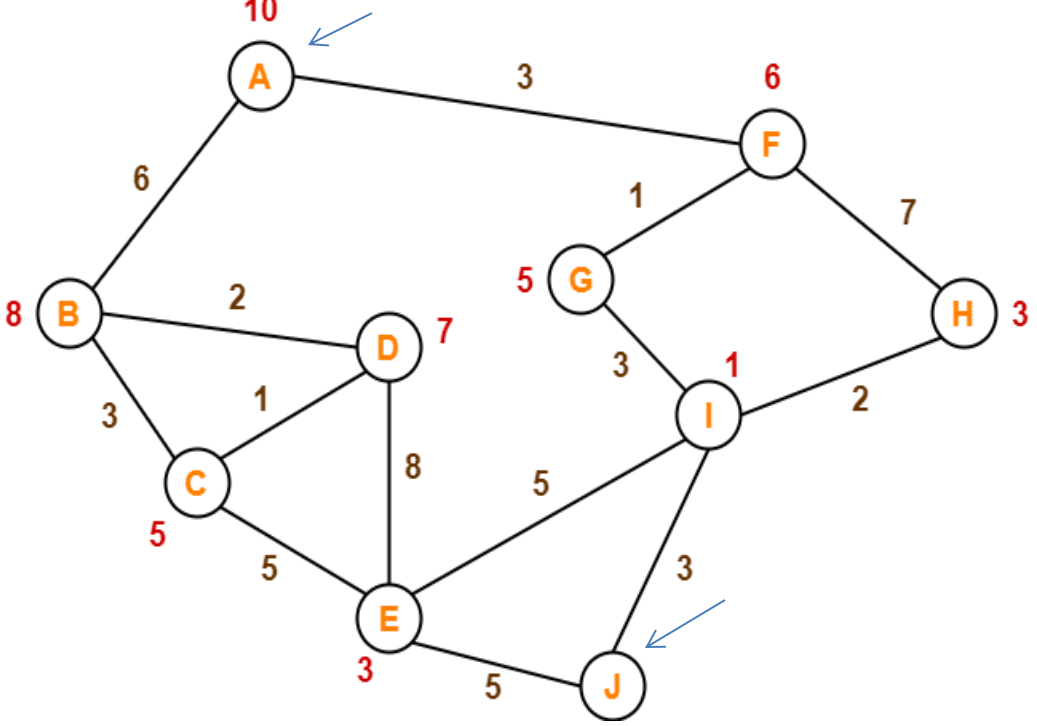
- We start with node A.
- Node B and Node F can be reached from node A.

A* Algorithm calculates f(B) and f(F).

- f(B) = 6 + 8 = 14
- f(F) = 3 + 6 = 9

Since f(F) < f(B), so it decides to go to node F.

Path- A → F



Step-02:

Node G and Node H can be reached from node F.

A* Algorithm calculates f(G) and f(H).

- f(G) = (3+1) + 5 = 9
- f(H) = (3+7) + 3 = 13

Since f(G) < f(H), so it decides to go to node G.

Path- A → F → G

Step-03:

Node I can be reached from node G.

A* Algorithm calculates f(I).

f(I) = (3+1+3) + 1 = 8

It decides to go to node I.

Path- A → F → G → I

Step-04:

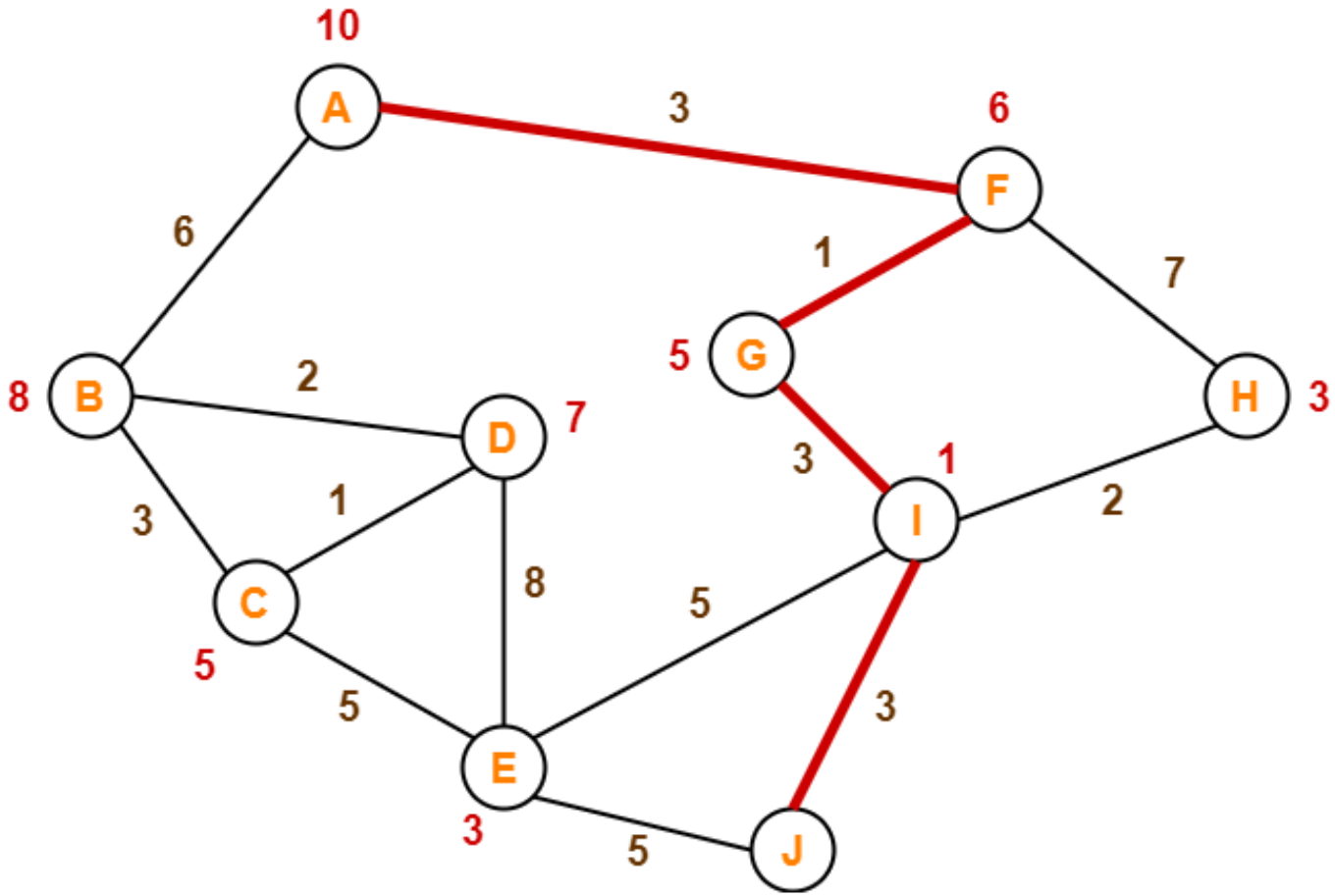
Node E, Node H and Node J can be reached from node I.

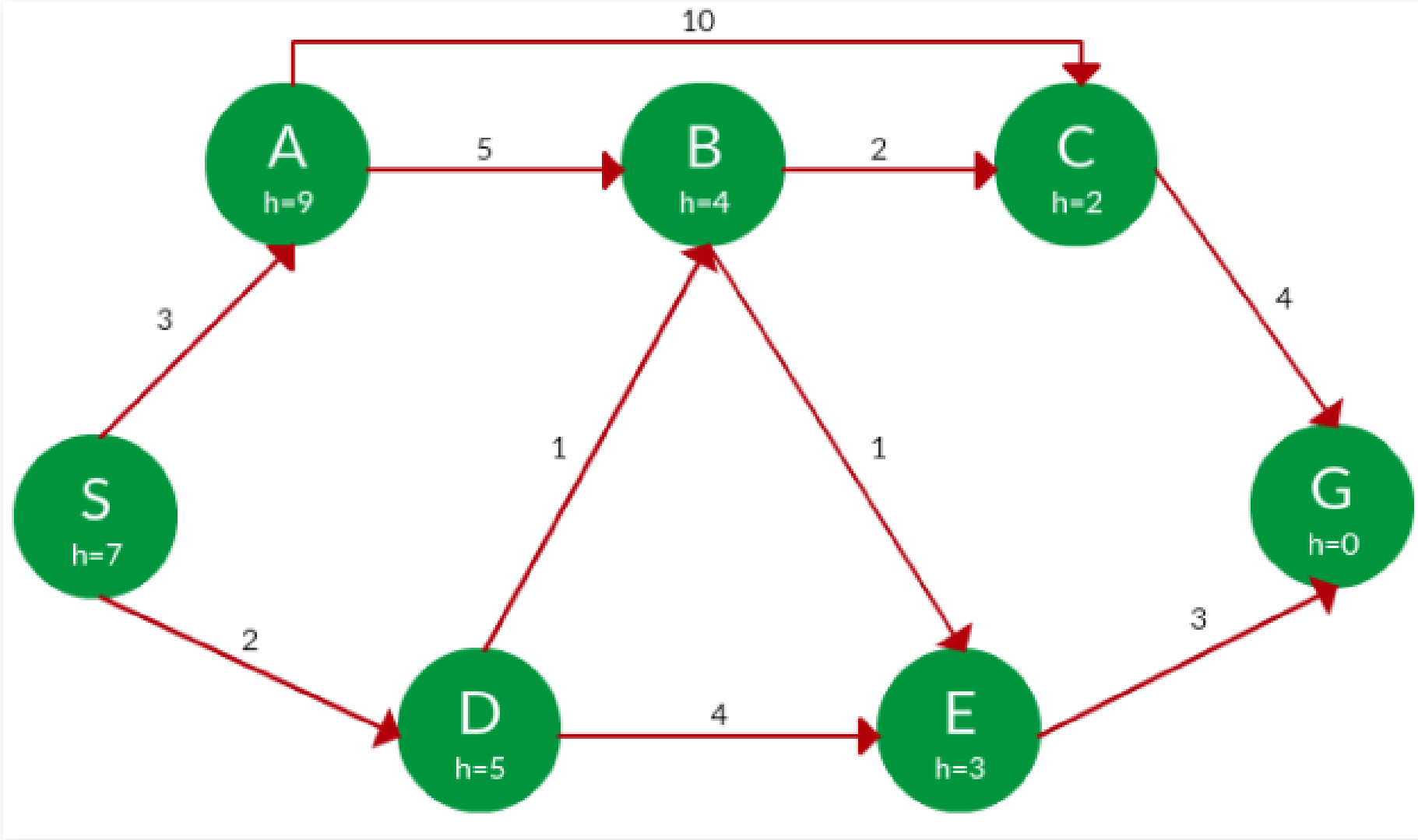
A* Algorithm calculates f(E), f(H) and f(J).

- f(E) = (3+1+3+5) + 3 = 15
- f(H) = (3+1+3+2) + 3 = 12
- f(J) = (3+1+3+3) + 0 = 10

Since f(J) is least, so it decides to go to node J.

Path- A → F → G → I → J

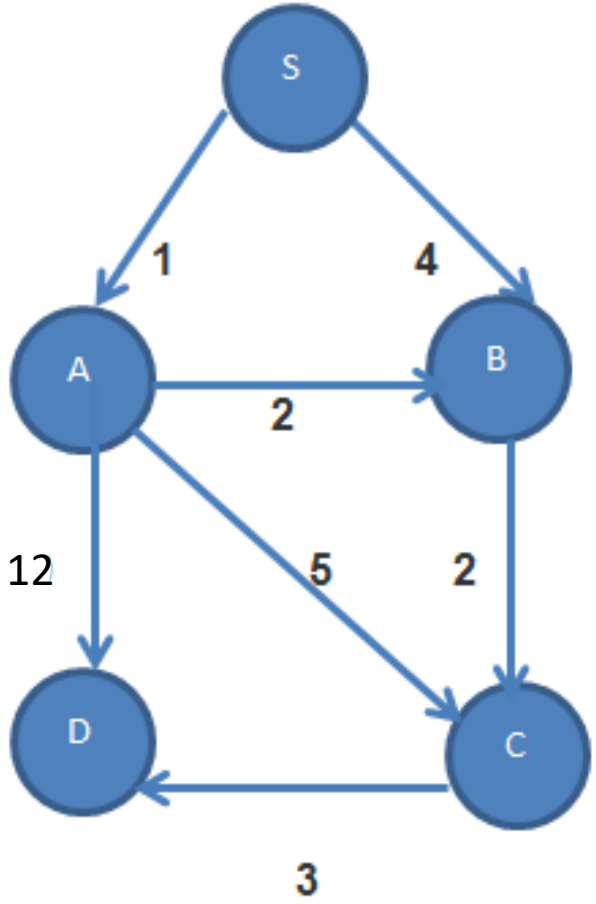

10
A
3
6
F
6
1
7
5
G
H
3
8
B
2
D
7
3
1
3
1
I
2
C
8
5
5
5
3
E
3
5
J


10
A
h=9
5
B
h=4
2
C
h=2
3
4
S
h=7
1
1
G
h=0
2
3
D
h=5
4
E
h=3

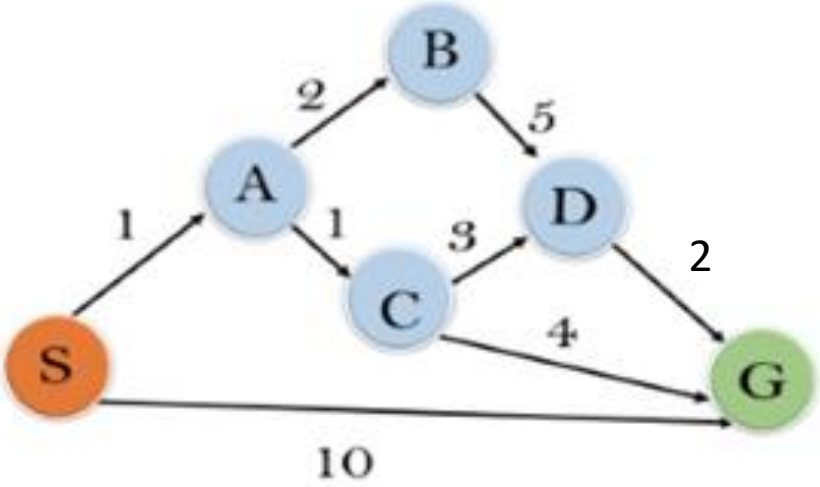
PATH	H(X)	G(X)	F(X)
S	7	0	7
S -> A	9	3	12
S -> D ✓	5	2	7
S -> D -> B ✓	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C ✓	2	3 + 2 = 5	7
S -> D -> B -> E ✓	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G ✓	0	4 + 3 = 7	7

Path: S -> D -> B -> E -> G

Cost: 7



S	7
A	6
B	2
C	1
D	0



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

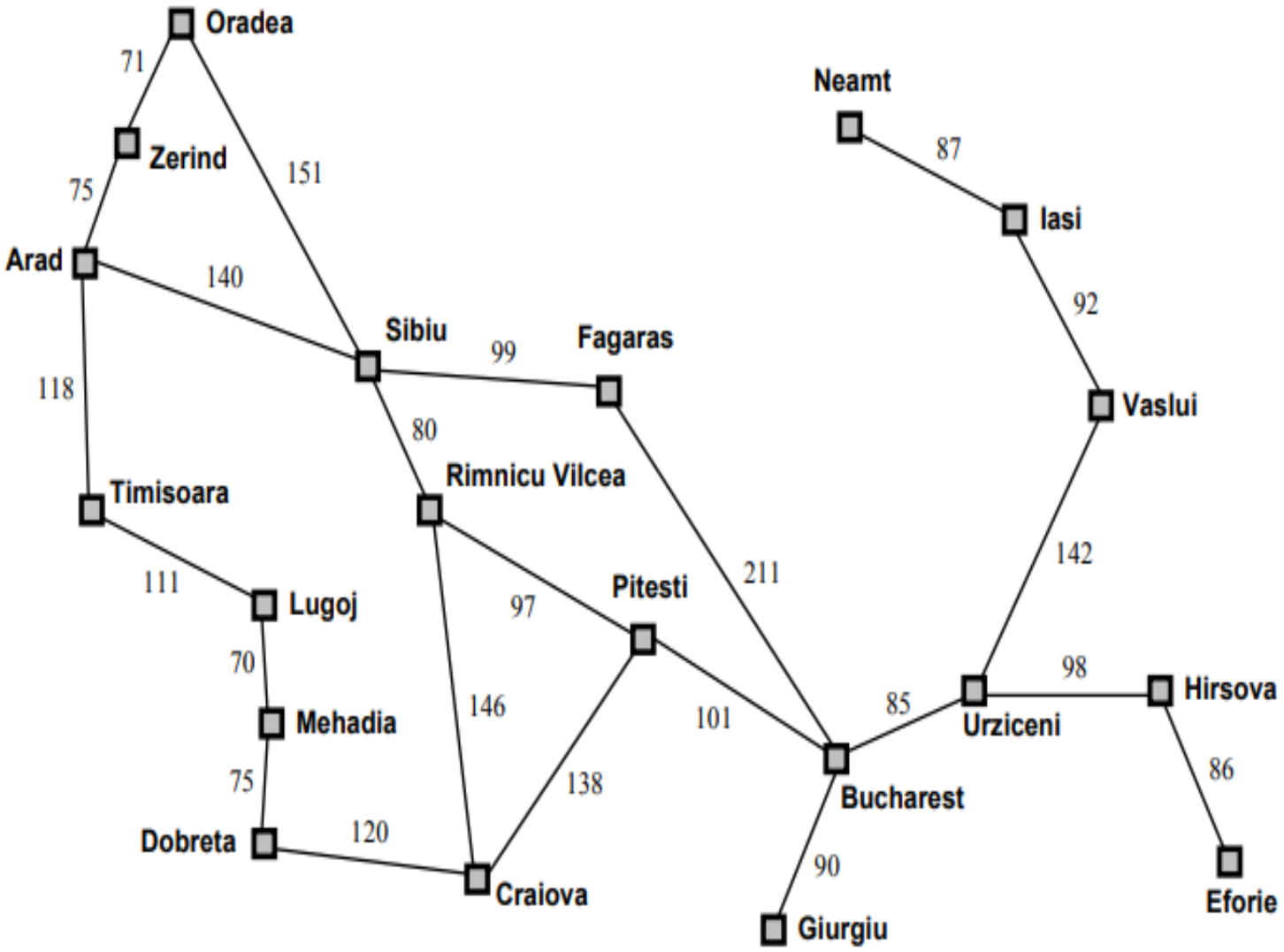
Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

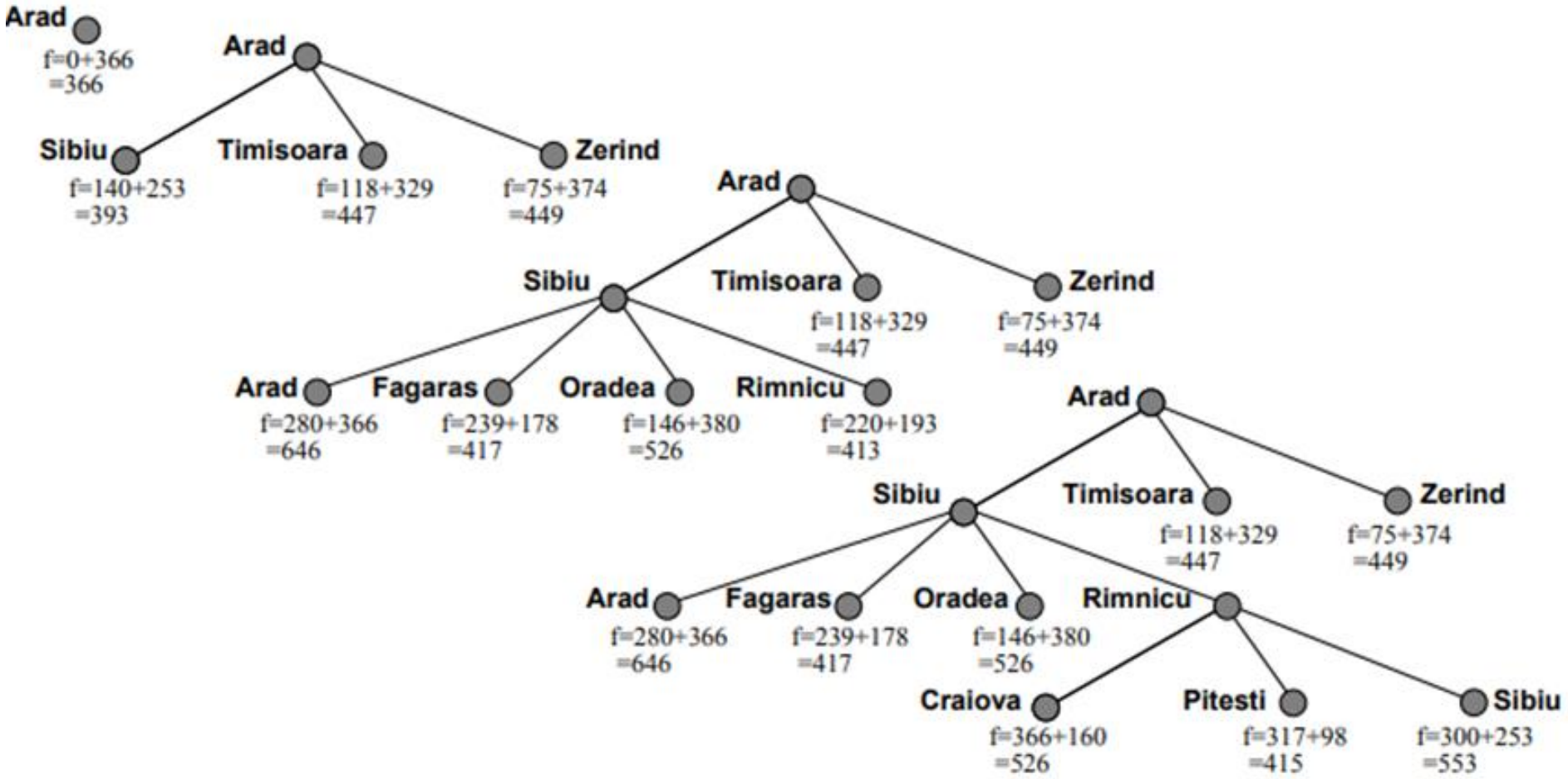
Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374


Arad
f=0+366
=366
Arad
Sibiu
f=140+253
=393
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
f=220+193
=413
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
Craiova
f=366+160
=526
Pitesti
f=317+98
=415
Sibiu
f=300+253
=553

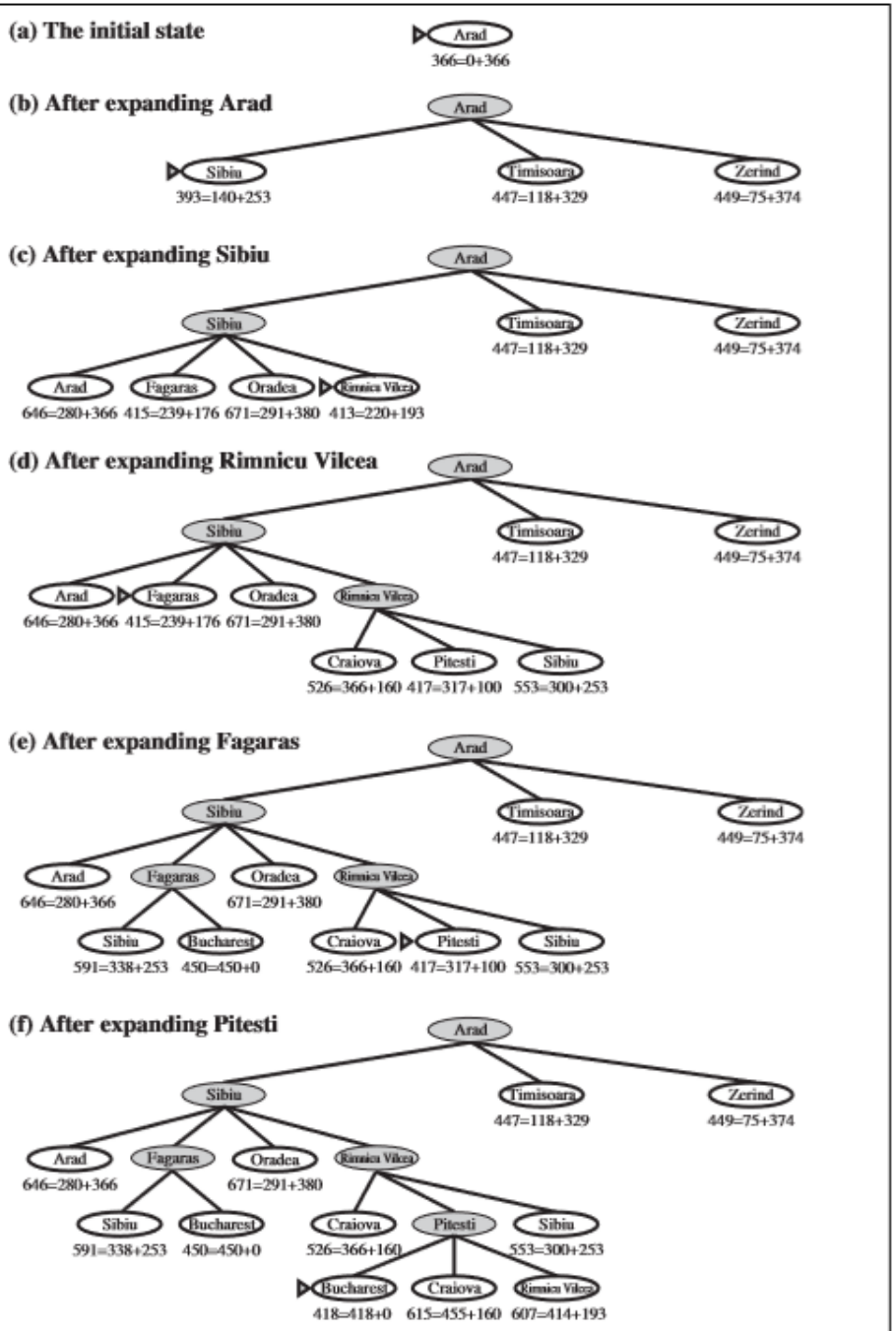


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

Advantages:

- **Better algorithm** than other search algorithms.
- **Optimal and complete**.
- Can solve very **complex problems**.

Disadvantages:

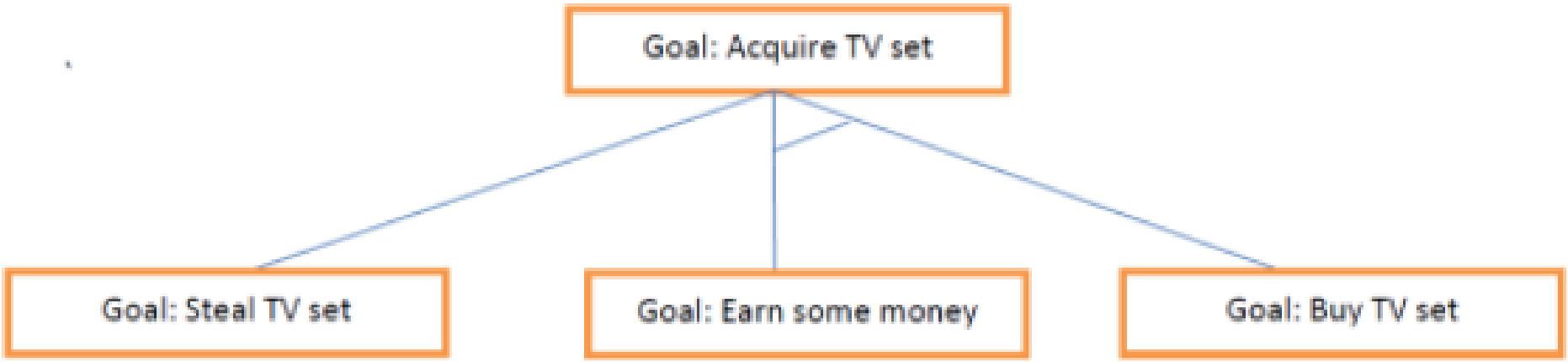
- It **does not always produce the shortest path** as it mostly based on heuristics and approximation.
- **Memory requirement** as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.
- **APPLICATIONS**: Traffic Navigational Sys, Underwater obstacle avoidance for unmanned surface vehicle (USV) ,Path finding technique,etc..

AND OR Graphs AO*

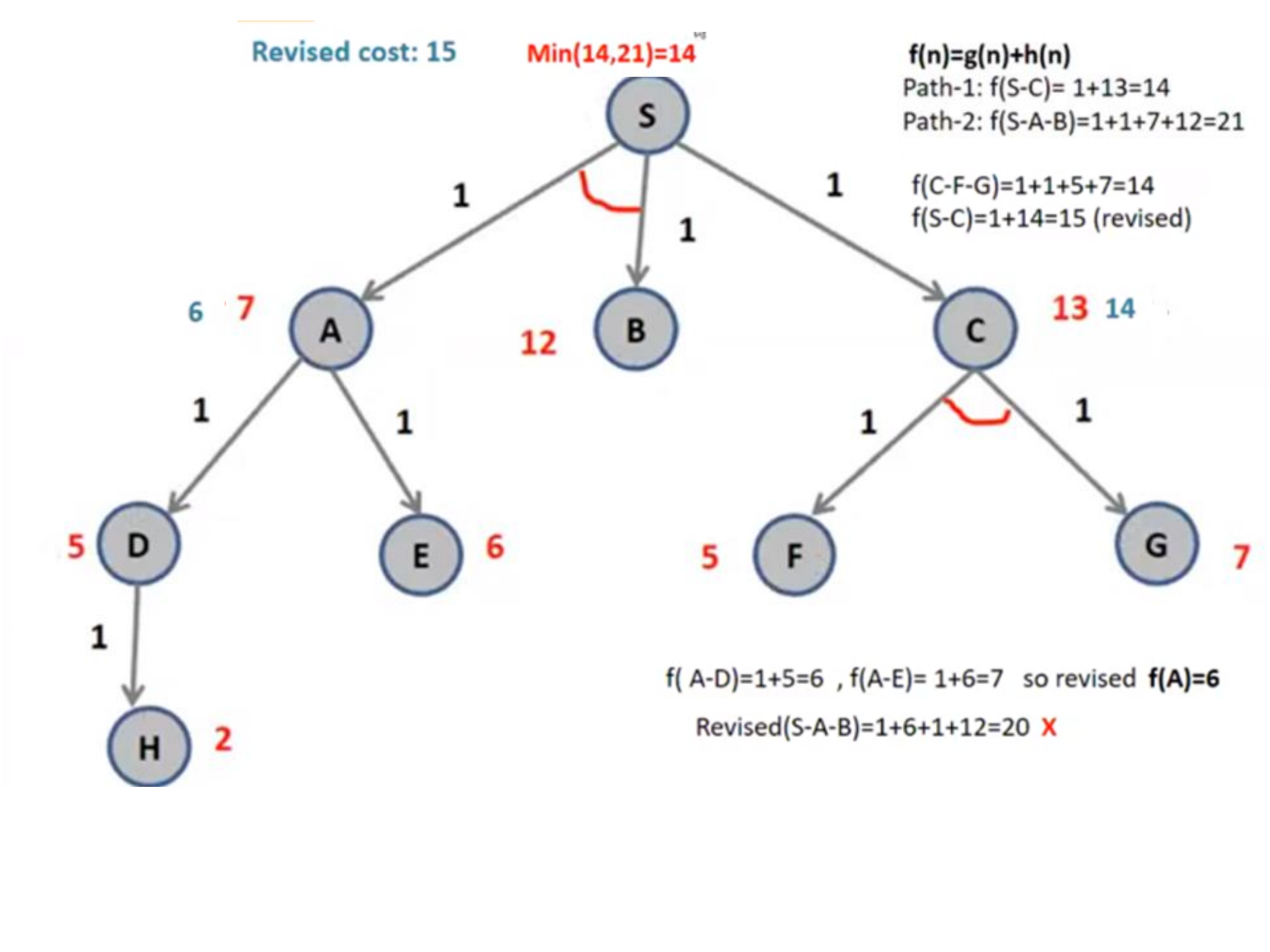
AO* is used to perform a heuristic search of an AND-OR graph.

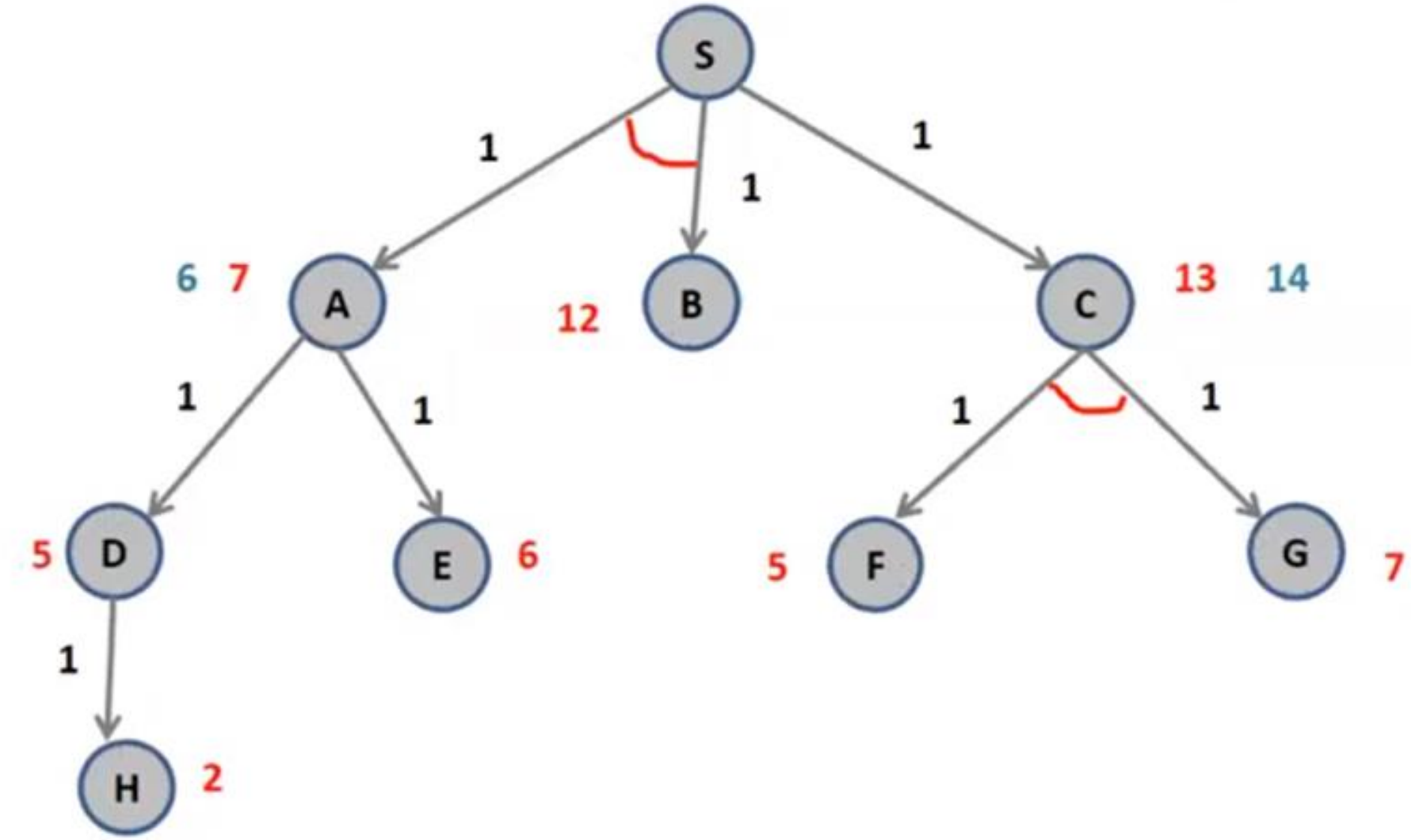
AO* Algorithm

- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution → AND-OR graphs.
  - AND - OR trees are used for representing the solution.
- The decomposition of the problem or problem reduction generates AND arcs.



A SIMPLE AND-OR GRAPH


Revised cost: 15
Min(14,21)=14
f(n)=g(n)+h(n)
Path-1: f(S-C)= 1+13=14
Path-2: f(S-A-B)=1+1+7+12=21
f(C-F-G)=1+1+5+7=14
f(S-C)=1+14=15 (revised)
S
A
B
C
D
E
F
G
H
1
6
7
12
13
14
5
6
5
7
2
f(A-D)=1+5=6 , f(A-E)= 1+6=7 so revised f(A)=6
Revised(S-A-B)=1+6+1+12=20 X


S
1
1
1
6
7
A
12
B
C
13
14
1
1
1
1
5
D
E
6
5
F
G
7
1
H
2

AO*: Example #1

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

Path-1: f(S-C)= 1+13=14

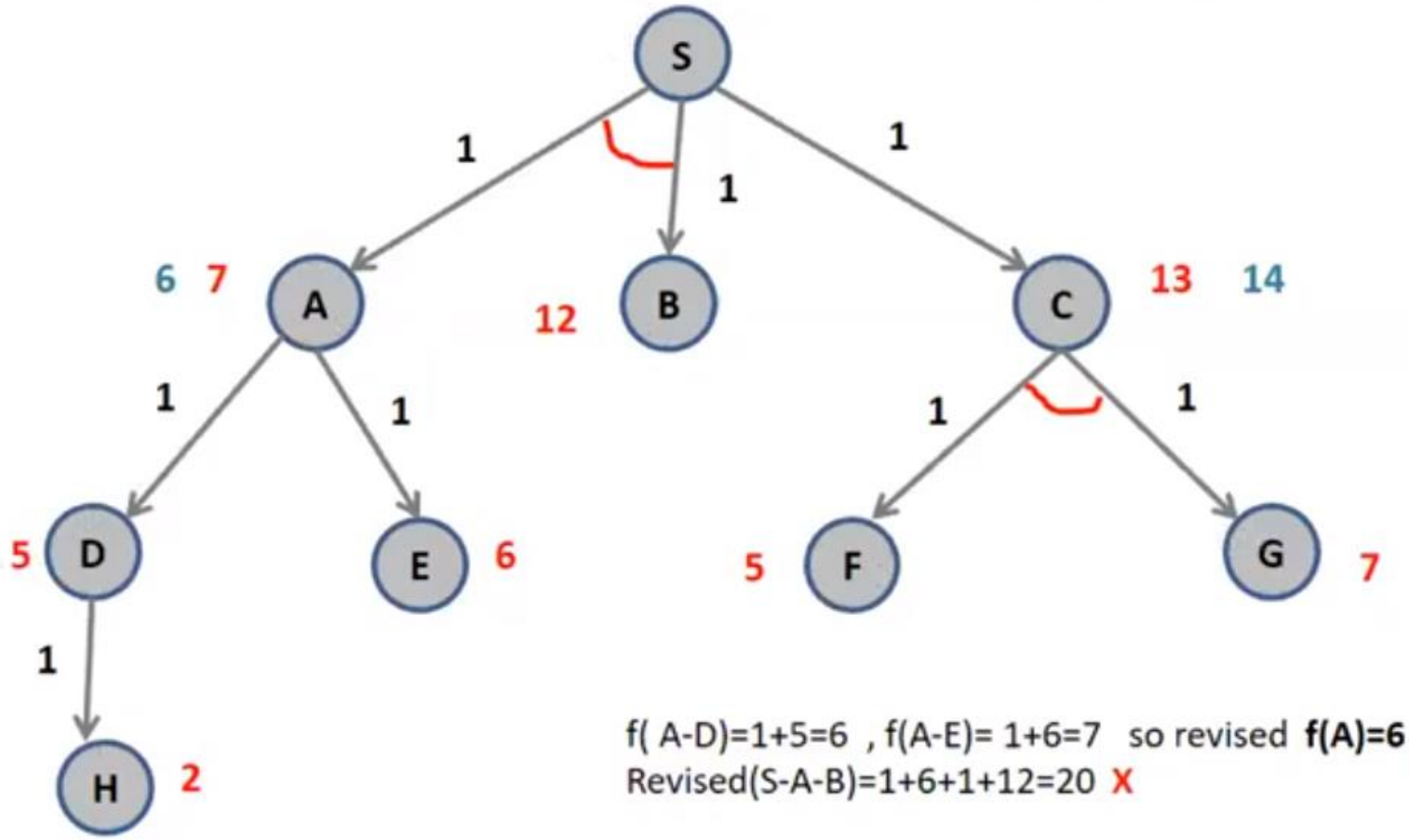
Path-2: f(S-A-B)=1+1+7+12=21

f(C-F-G)=1+1+5+7=14

f(S-C)=1+14=15 (revised)

Revised cost: 15

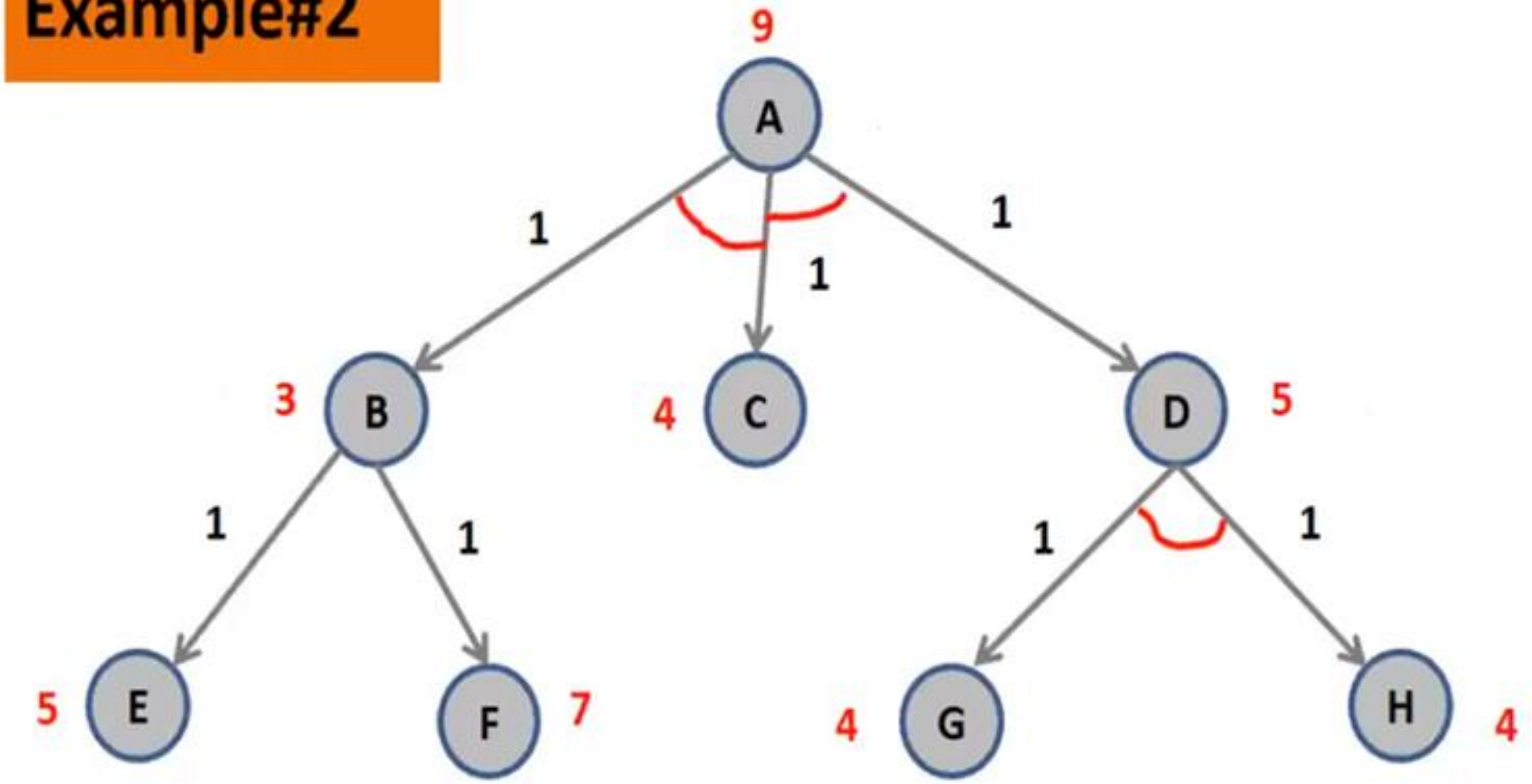
Min(14,21)=14



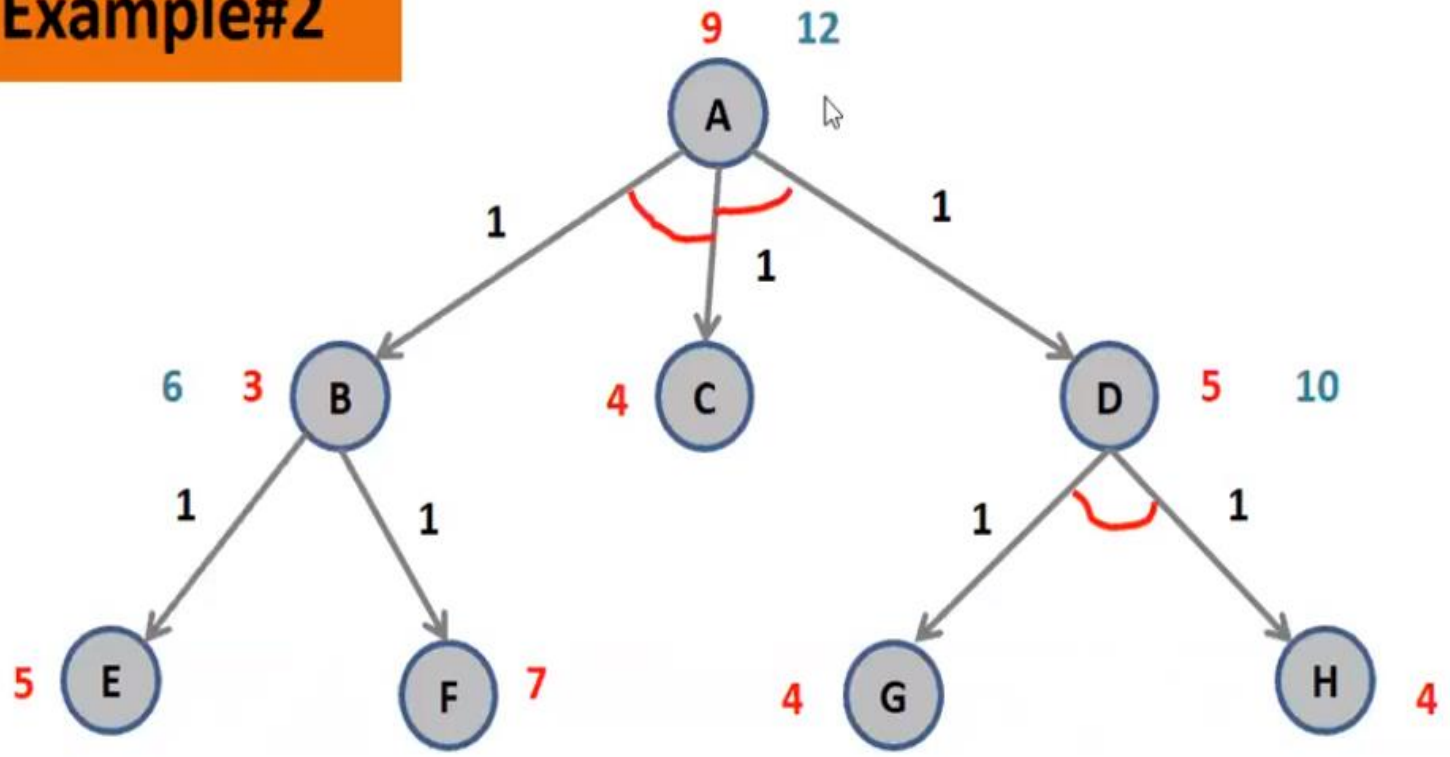
f(A-D)=1+5=6 , f(A-E)= 1+6=7 so revised **f(A)=6**

Revised(S-A-B)=1+6+1+12=20 X

Example#2

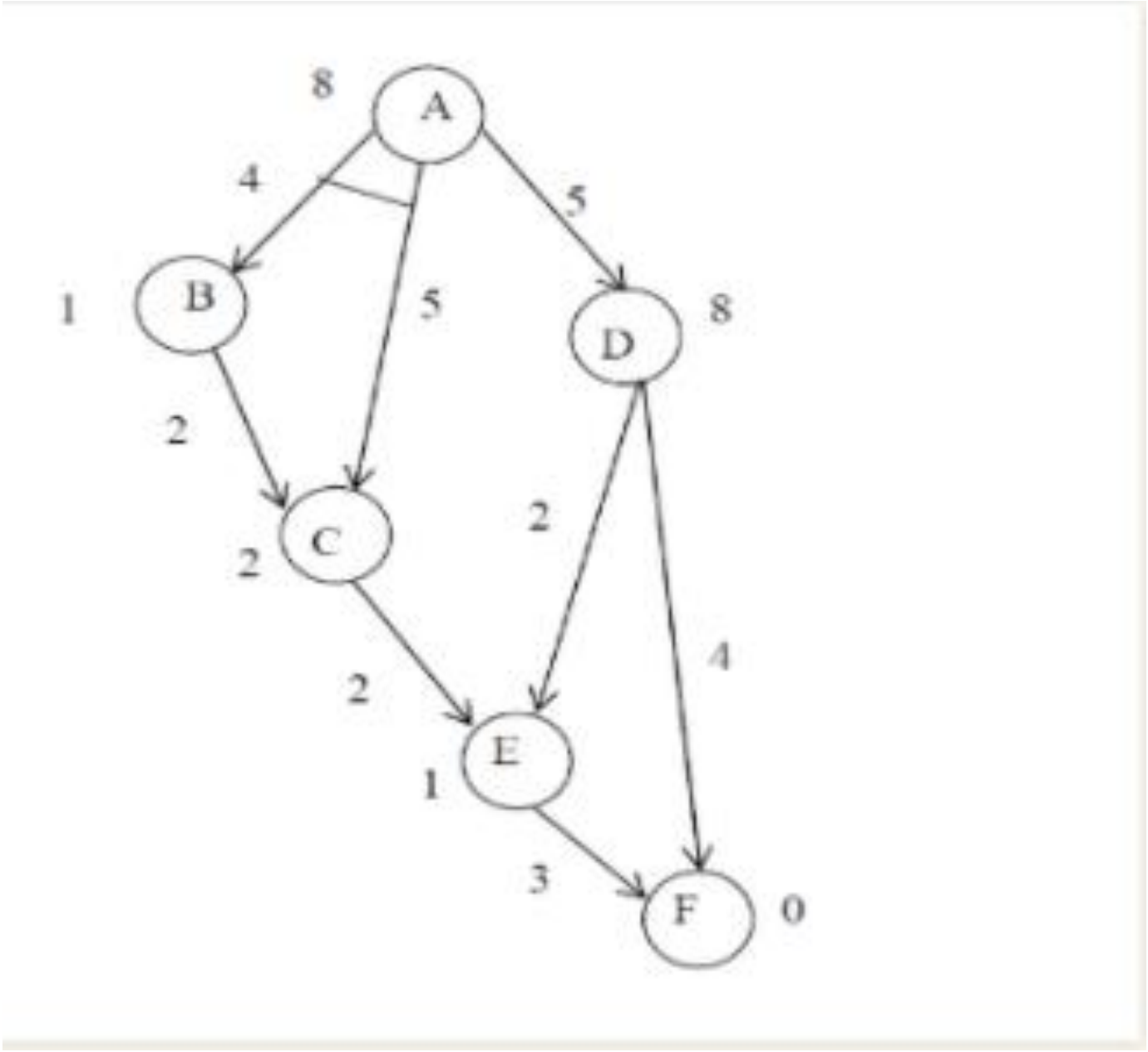


Example#2

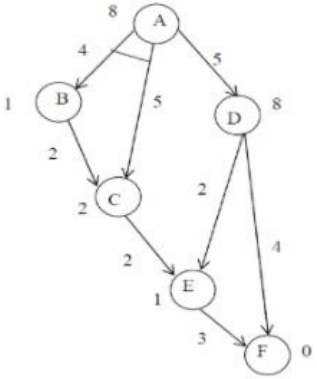


Path -1: f(A-B-C)= 1+1+3+4= 9
f(B-E)=1+5=6 f(B-F)=1+7=8
f(A-B-C)= 1+1+6+4= 12

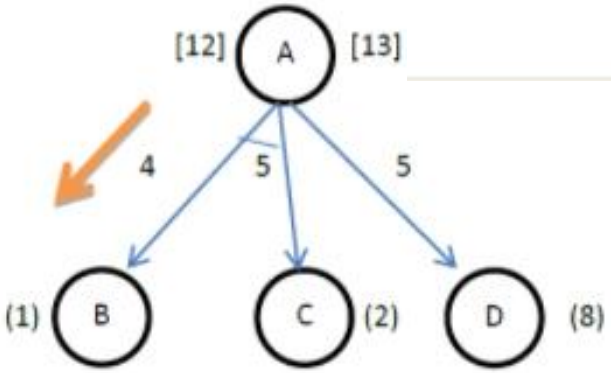
Path-2: f(A-C-D)=1+1+4+5=11
f(D-G-H)= 1+1+4+4 =10
f(A-C-D)= 1+1+4+10= 16

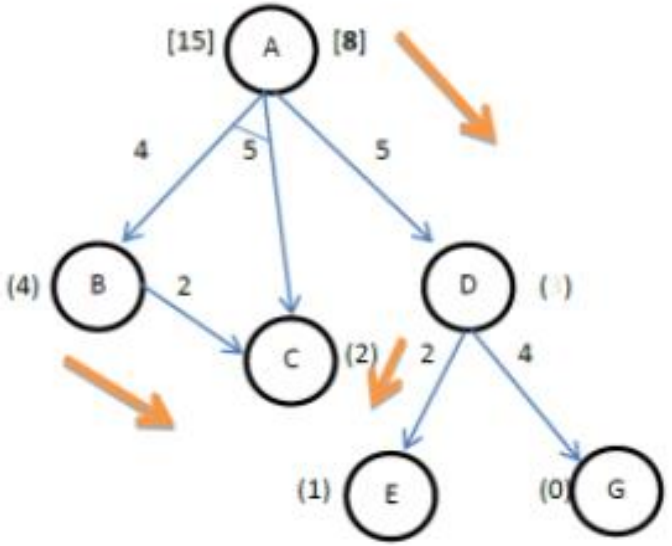

8
A
4
5
1
B
5
D
8
2
2
C
2
2
4
2
E
1
3
F
0

STEP1:



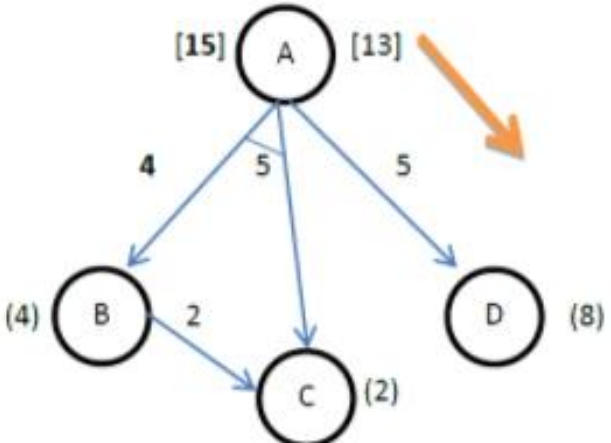
Step2:

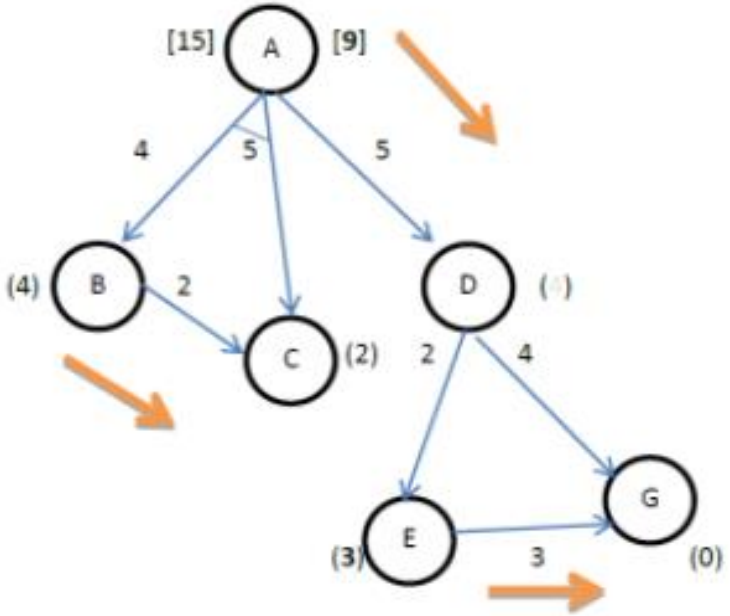




STEP4:

STEP3:





Conditions for optimality

ADMISSIBILITY AND CONSISTENCY

1) h(n) be an admissible heuristic.

- **An admissible heuristic is one that never overestimates the cost to reach the goal.**

2) g(n) is the actual cost to reach “n” along the current path, and f(n)=g(n)+h(n),

i.e f(n) never overestimates the true cost of a solution along the current path through n.

Ex: Straight line distance

Admissibility of the heuristic function is given as:

$h(n) <= h^*(n)$

h(n) is heuristic cost, and

h*(n) is the estimated cost.

"Hence heuristic cost should be less than or equal to the estimated cost."

- Slightly stronger condition called – **CONSISTENCY**

heuristic **h(n) is consistent** if,

- for every node n and every successor n' of n

generated by any action a,

The estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$h(n) \leq c(n, a, n') + h(n')$.

- Heuristics are "rules of thumb", educated guesses, intuitive judgments or simply common sense.

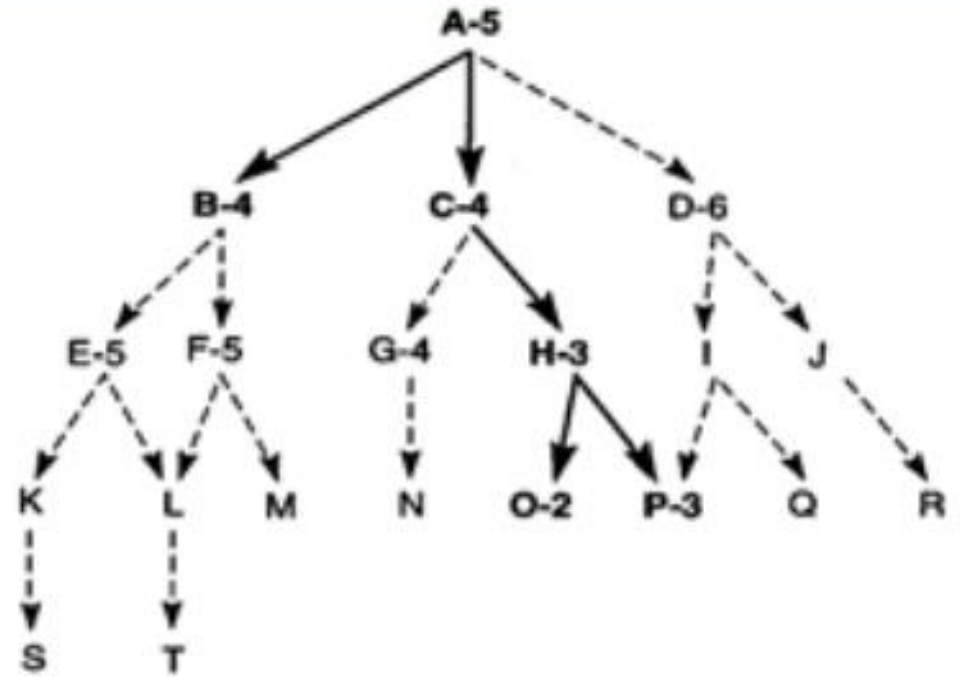


Figure 4.4 Heuristic search of a hypothetical state space.

Admissible Heuristics

- Never over estimated
- Guaranteed to find optimal solutions
- Based on evaluation function: $f'(s) = h(s) + h'(s)$
-
  - *h'(s)*: estimate of length of shortest path from s to goal state
  - *h(s)*: actual length

Heuristics for 8 puzzle problem

STATE(N)

Goal state

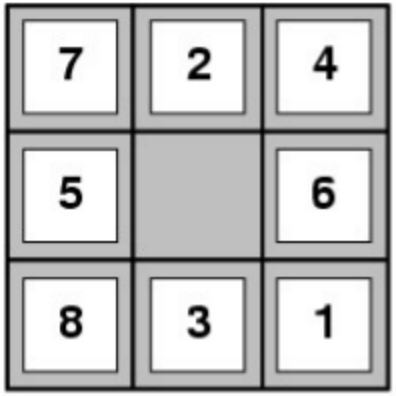
- $h_1(N)$ = number of misplaced numbered tiles = 6

$h_2(N)$ = sum of the (Manhattan) distance of every numbered tile to its goal position

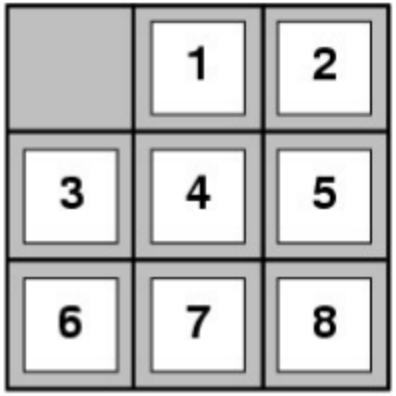
= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

Find Heuristics for each of the following 8 puzzle problem buy both techniques

Example: 8-puzzle



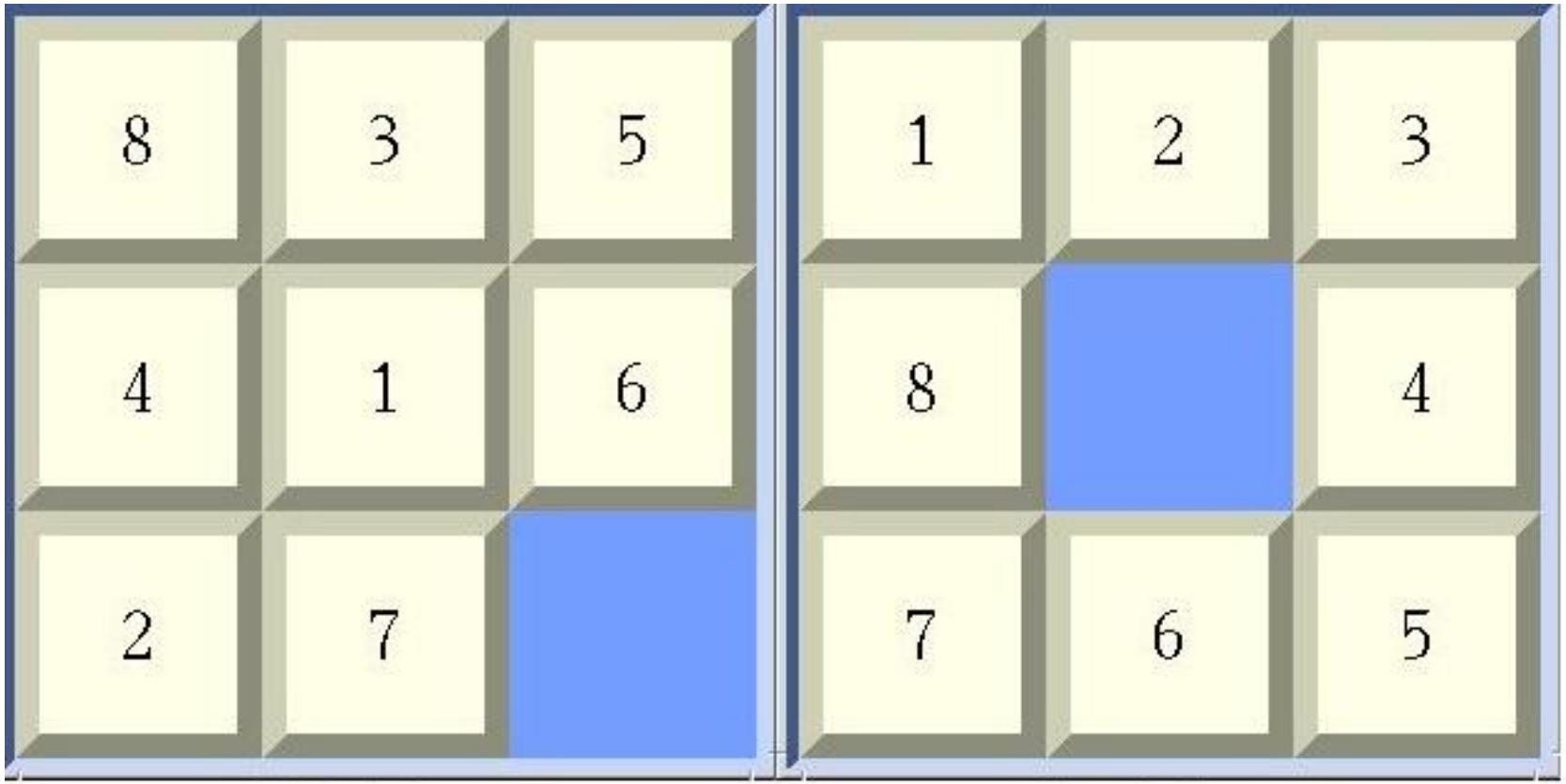
Start State



Goal State

NP-Complete

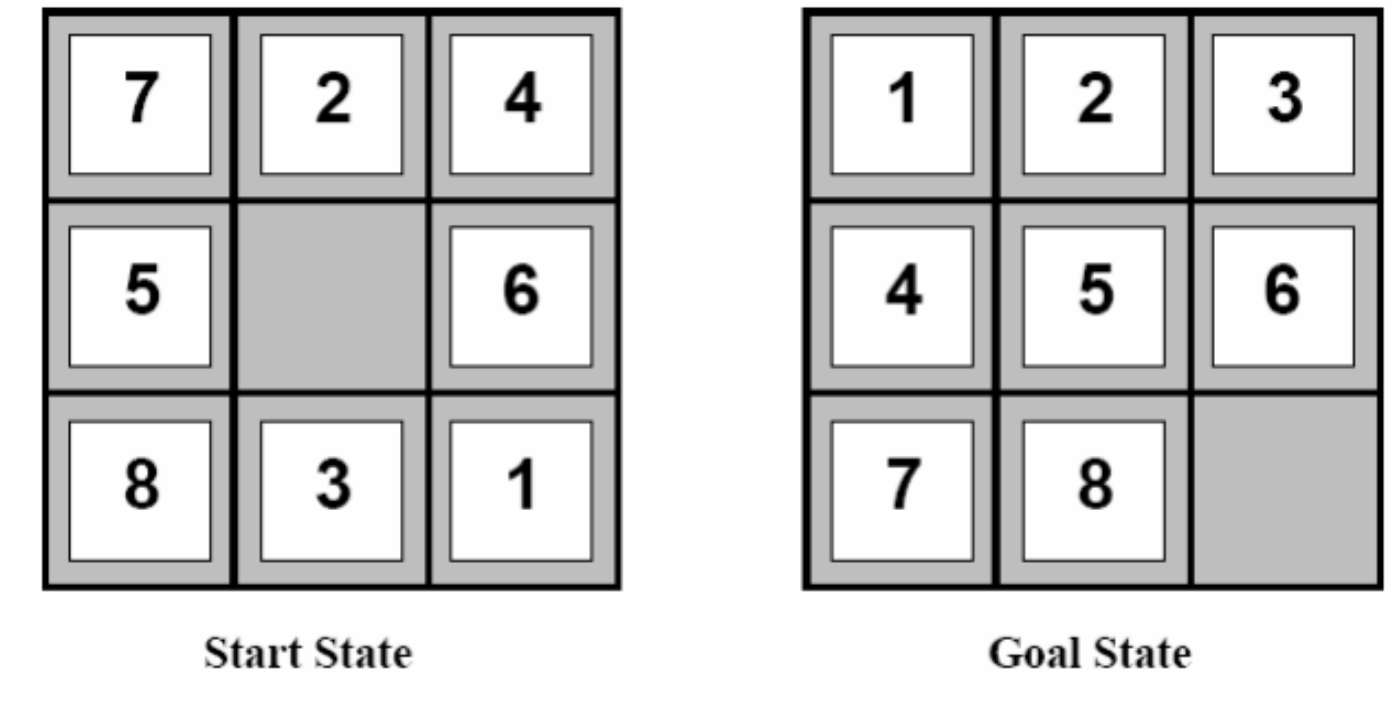
- States:
  - location of each tile and the blank
- Initial state: any, 9!/2
- Actions:
  - blank moves Left, Right, Up or Down
- Transition model:
  - Given a state and action, returns the resulting state
- Goal test: Goal configuration
- Path cost: Each step costs 1



Initial State

Goal State

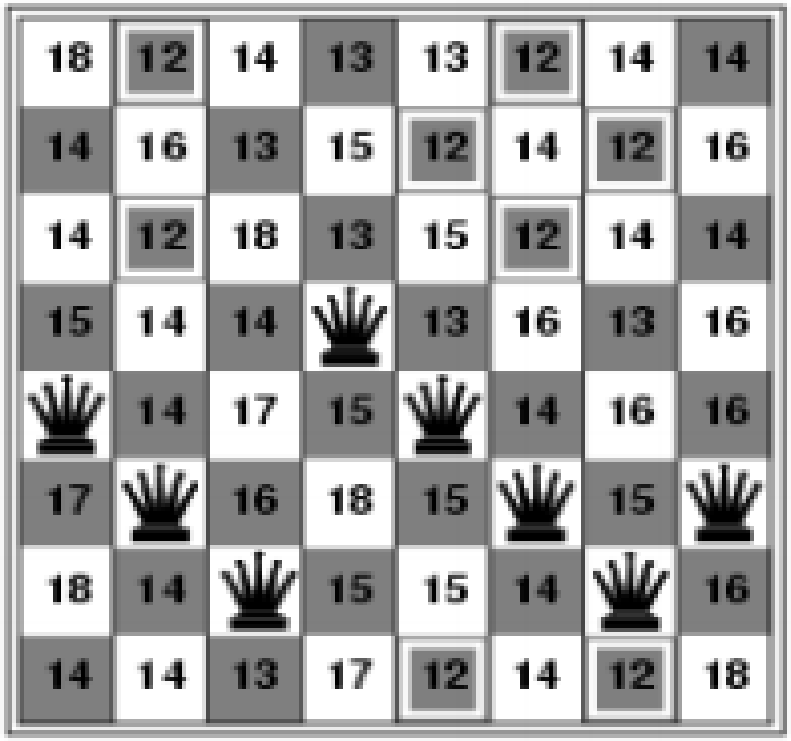
8-puzzle



Rule: Can slide a tile into the blank spot. (Equivalently, can think if it as moving the blank around).

Heuristics for 8 queen problem

Heuristic?



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

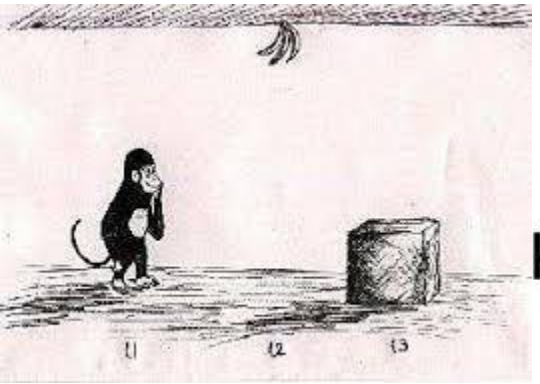
- A hungry monkey is in a room.
- Bananas have been hung from the center of the ceiling of the room.
- In the corner of the room there is a chair.
- The monkey wants the bananas but he can't reach them.
- What shall he do?

If the monkey is cleaver enough, he can reach the bananas by placing the chair directly below the bananas and climbing on the top of the chair.

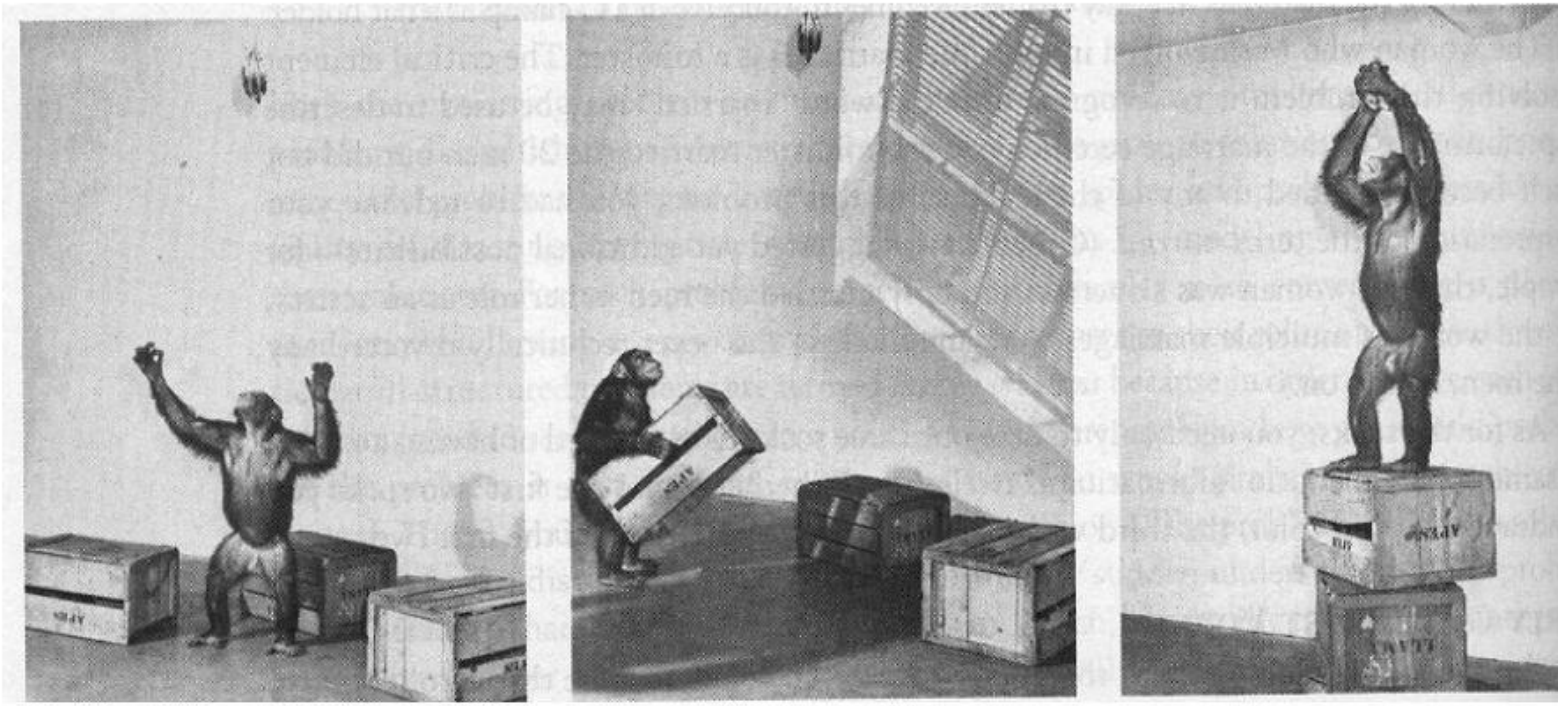
Monkey Banana Problem

There is a monkey at the door into a room. Inside the room at the middle a banana is hanging from the ceiling and there is box at the window which can be used by monkey to grasp the banana. Monkey is allowed following actions.

- 1. Walk on the floor
- 2. Climb the box
- 3. Push the box around (if it is already at the box)
- 4. Grasp the banana if standing on the box directly under the banana.



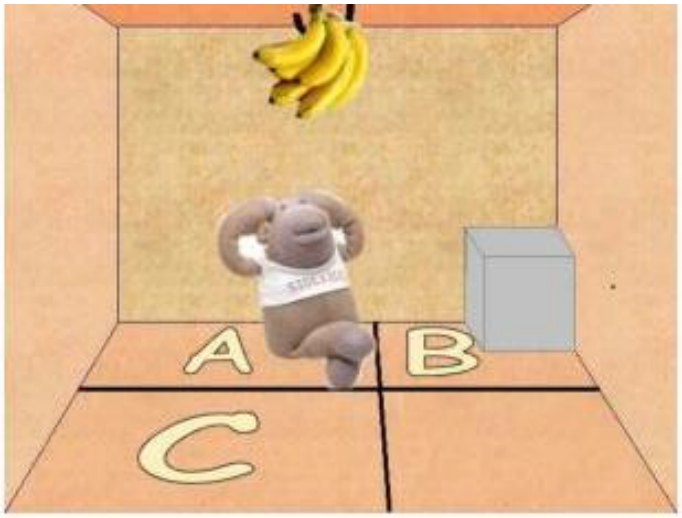
Kohler (1945): monkey and banana problem.



Kohler observed that chimpanzees appeared to have an insight into the problem before solving it

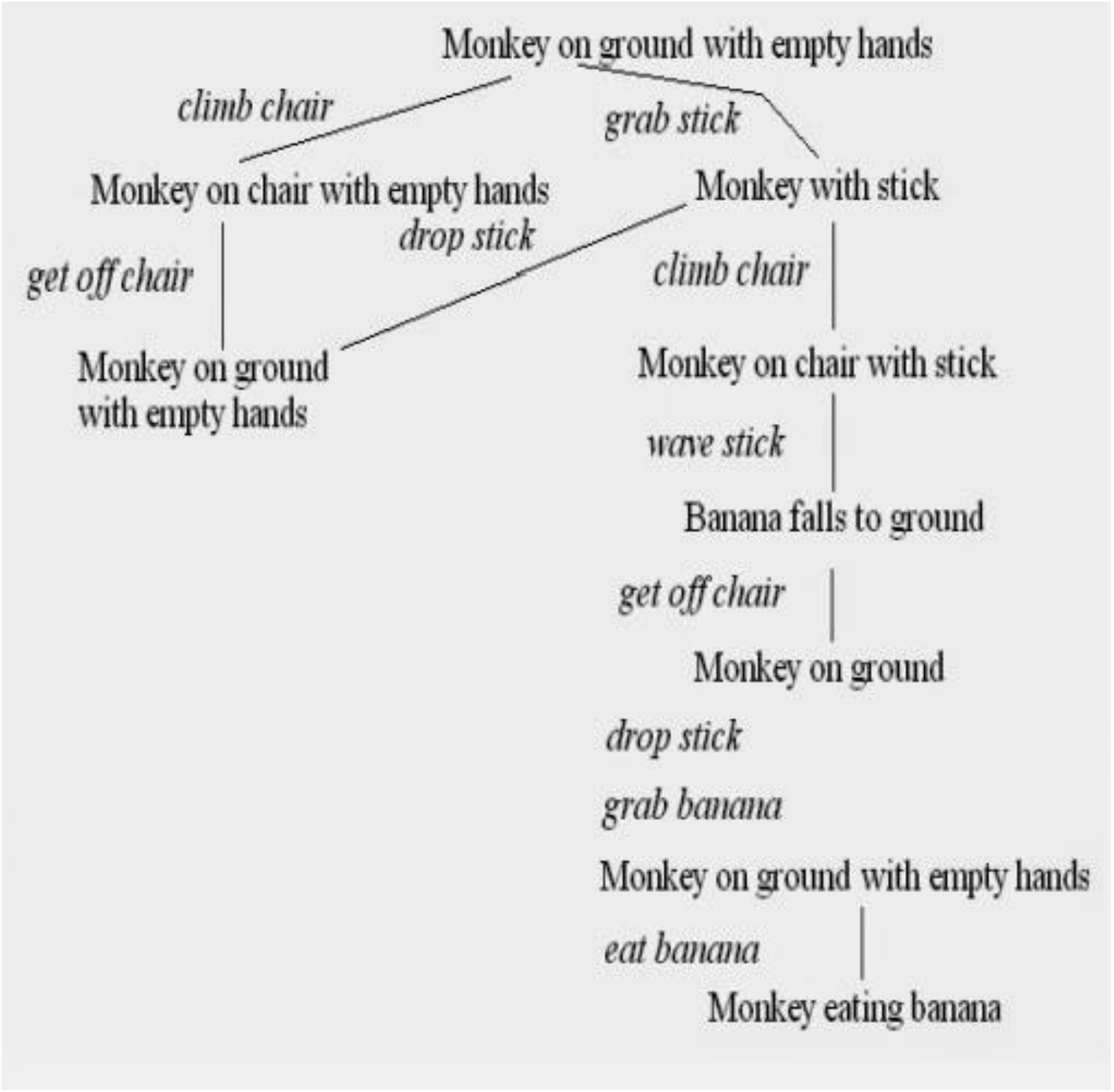
Initial State:

```
on(monkey, floor),
on(box, floor),
at(monkey, a),
at(box, b),
at(bananas, c),
status(bananas, hanging).
```



Goal State:

```
on(monkey, box),
on(box, floor),
at(monkey, c),
at(box, c),
at(bananas, c),
status(bananas, grabbed).
```


Monkey on ground with empty hands
climb chair
grab stick
Monkey on chair with empty hands
Monkey with stick
drop stick
get off chair
climb chair
Monkey on ground
with empty hands
Monkey on chair with stick
wave stick
Banana falls to ground
get off chair
Monkey on ground
drop stick
grab banana
Monkey on ground with empty hands
eat banana
Monkey eating banana

Local Search Algorithms

- **Local search** is a **heuristic** method for solving computationally hard optimization problems.
- Moves from solution to solution in the space of candidate solutions (the **search** space) by applying **local** changes, until a solution deemed optimal is found or a time bound is elapsed.

LOCAL SEARCH ALGORITHMS

"IF THE PATH TO THE GOAL DOES NOT MATTER, WE MIGHT CONSIDER A DIFFERENT CLASS OF ALGORITHMS, ONES THAT DO NOT WORRY ABOUT PATHS AT ALL. "

- Local search algorithms operate using a **single current node** (rather than multiple paths) and generally **move only to neighbours of that node.**
  - Typically, the paths followed by the search are **not retained**

- Although local search algorithms are **not systematic**, they have two key advantages:
  - (1) they use very **little memory**—usually a constant amount; and
  - (2) they can often **find reasonable solutions in large or infinite (continuous)** state spaces for which systematic algorithms are unsuitable

Applications

Local search algorithms are widely applied to numerous hard computational problems, including problems from

- computer science (particularly **artificial intelligence**),
- mathematics,
- operations research,
- engineering, and
- bioinformatics.

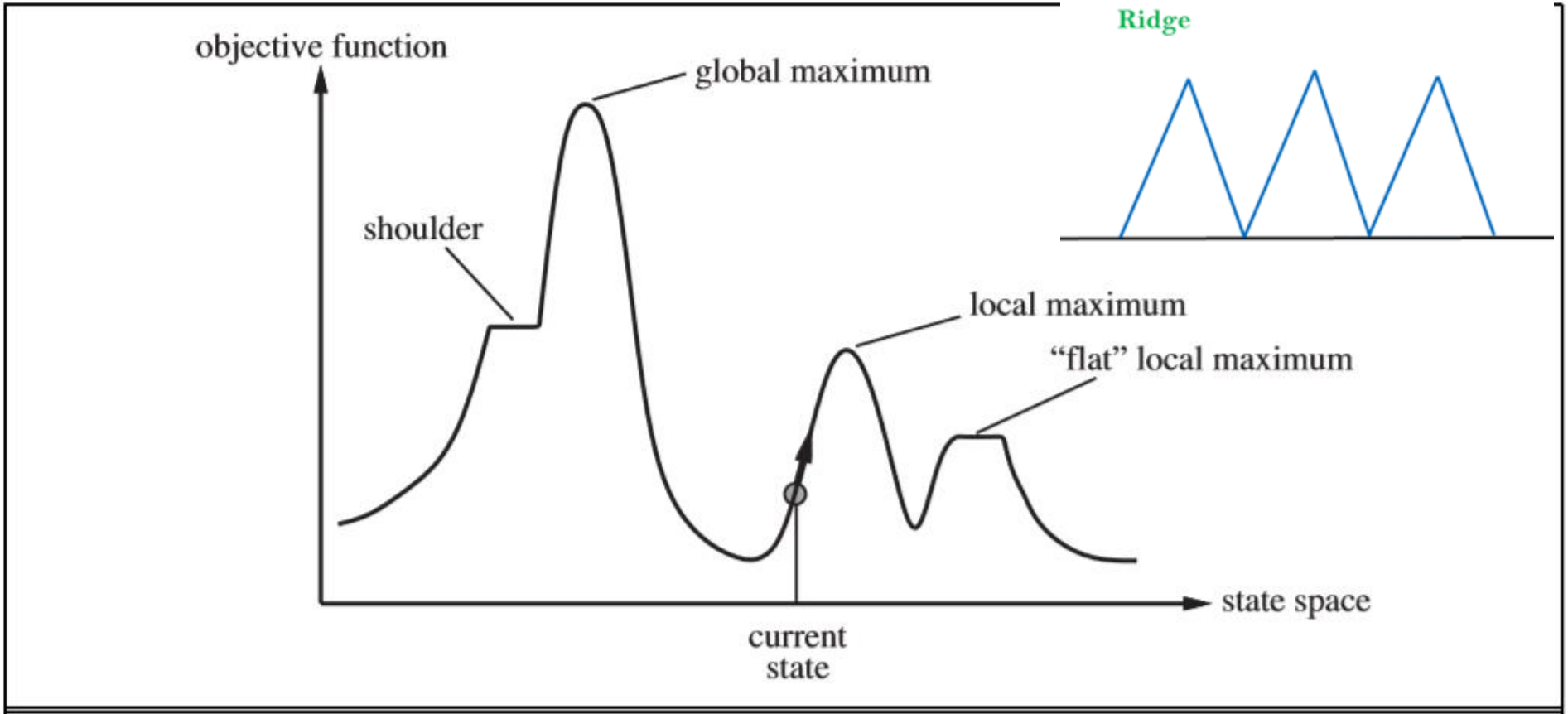


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

STATE-SPACE LANDSCAPE:

A landscape has both "location" (defined by the **state**) and " elevation" (defined by the value of the **heuristic COST** function or **OBJECTIVE FUNCTION**).

Different regions in the State Space Diagram

- **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum).
  - This state is better because here the value of the objective function is higher than its neighbors.
- **Global maximum :** It is the best possible state in the state space diagram.
  - This because at this state, objective function has highest value.
- **Plateau/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
- **Ridge :** It is region which is higher than its neighbours but itself has a slope.
  - It is a special kind of local maximum.
- **Current state :** The region of state space diagram where we are currently present during the search.
- **Shoulder :** It is a plateau that has an uphill edge

HILL CLIMBING PROBLEM

HILL CLIMBING PROBLEM

1.Define the current state as an initial state

2.Loop until the goal state is achieved or no more operators can be applied on the current state:

1. Apply an operation to current state and **get a new state**
2. **Compare** the new state with the goal
3. **Quit** if the goal state is achieved
4. Evaluate new state with heuristic function and **compare it with the current state**
5. **If the newer state is closer** to the goal compared to current state, **update the current state**

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

- The hill-climbing search algorithm (steepest-ascent version) is simply a **loop that continually moves in the direction of increasing value**—that is, uphill.
- It **terminates** when it reaches a “peak” where no neighbour has a higher value.
- D**oes not maintain a search tree**, so the data structure for the current node needs only to record the state and the **value of the objective function.**
- Hill climbing does not look ahead beyond the immediate neighbours of the current state.

This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia!!!

Disadvantages of Hill Climbing

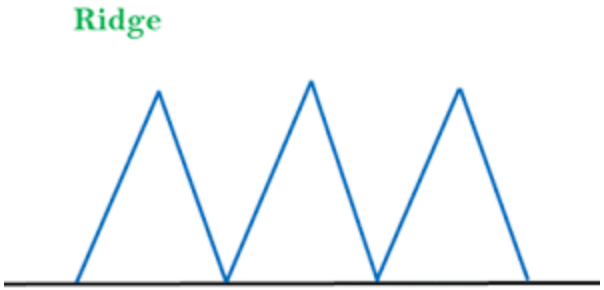
1) **LOCAL MAXIMA:**

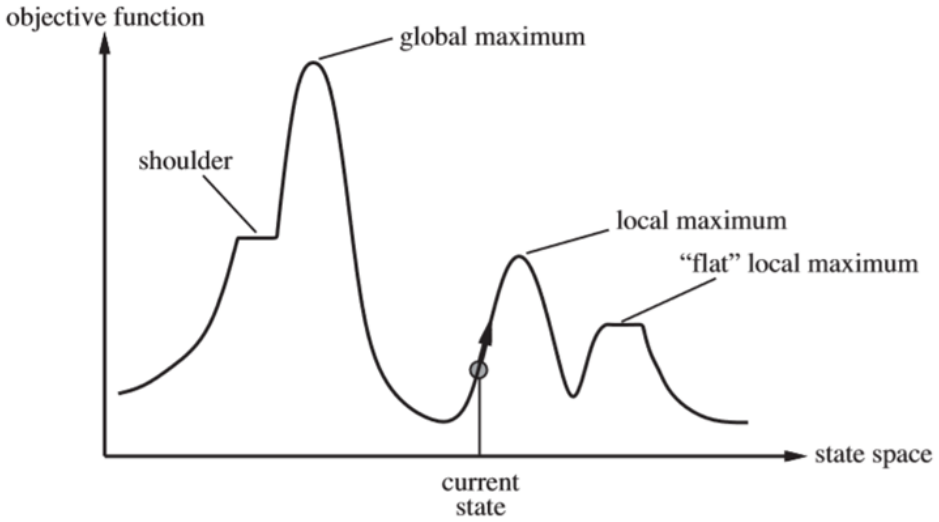
A local maximum is a **peak that is higher than each of its neighbouring states but lower than the global maximum**

- Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.

- 2) **RIDGES**: Ridges result in a **sequence of local maxima** that is very difficult for greedy algorithms to navigate.

- 3) **PLATEAUX**: a plateau is a flat area of the state-space landscape.
    - It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible
    - A hill-climbing search might get lost on the plateau.





Constraint Satisfaction Problem

- Goal → Discover some problem state that satisfies a given set of constraints
  - Eg: Cryptarithmetic Puzzles,
  - Eg: Design Tasks-create a design in fixed limits on time,cost and materials
- CSP operates in space of constraint sets
  - **Initial State**-Constraints originally stated in prob description
  - **Goal state-** Any state that has been constrained enough
  - where "enough "must be defined in each problem
- CSP-2 step process
  - Constraints are discovered and propogated as far as possible throughout the system,else search begins
  - Hypothesis about a way to strenghten the constraints

ALGORITHM* Pg 69

- Constraint
  - No two letters have same value
  - Sum of digits must be as shown in the problem
  - There should be only one carry forward
  - Digits that can be assigned to a word/alphabet (0-9)

```
  T O
+ GO
-------
O U T  → T , O , G , U
```

SUDOKU

4							5	9
2	6		5				3	
				9	2			
		2		6			1	
		3	8	1	9	7		
	7			3		5		
			3	4				
	3				6		2	7
5	9							6

Puzzle

4	1	7	6	8	3	2	5	9
2	6	9	5	7	1	8	3	4
3	8	5	4	9	2	6	7	1
8	4	2	7	6	5	9	1	3
6	5	3	8	1	9	7	4	2
9	7	1	2	3	4	5	6	8
7	2	6	3	4	8	1	9	5
1	3	8	9	5	6	4	2	7
5	9	4	1	2	7	3	8	6

Solution

```
  T O
+ GO
-------
O U T
```

- O= 1 , assuming carry is generated.(9+9,8+7,etc) i.e **<u>Left most digit will always be 1</u>**
  - ```
      T   1
    +   G   1
        -----------
    1   U   T
    ```
  - T= 2 as O + O = 1+1=2
  - ```
      2   1
    +   G   1
        -----------
    1   U   2
    ```
- G + 2 = 10 + U.
- If G = 9, U = 1. Which is not valid since O = 1.
- So, G = 8 and U = 0.
- Hence, O + U + T = 1 + 0 + 2 = 3

CSP-Eg Crypt arithmetic Puzzle

```
   T O
+  G O
_______
O  U  T
```

Letter		Digit
T	--	2
O	--	1
G	---	8
U	----	0

Solve

TWO+TWO=FOUR


T W O
+ T W O
F O U R
7 6 5
+ 7 6 5
1 5 3 0

1) DAYS+TOO=SHORT

2) BASE + BALL = GAMES

3) SEND+MORE=MONEY

D A Y S

+ T O O

S H O R T

9 7 4 3

+ 5 2 2

---

1 0 2 6 5

that means:

S = 1,

H = 0,

O = 2,

R = 6,

T = 5

so, S + H + O + R + T = 1 + 0 + 2 + 6 + 5 = 14

B	7
A	4
S	8
E	3
L	5
G	1
M	9

```
  S E N D
+ M O R E
---------
M O N E Y
```

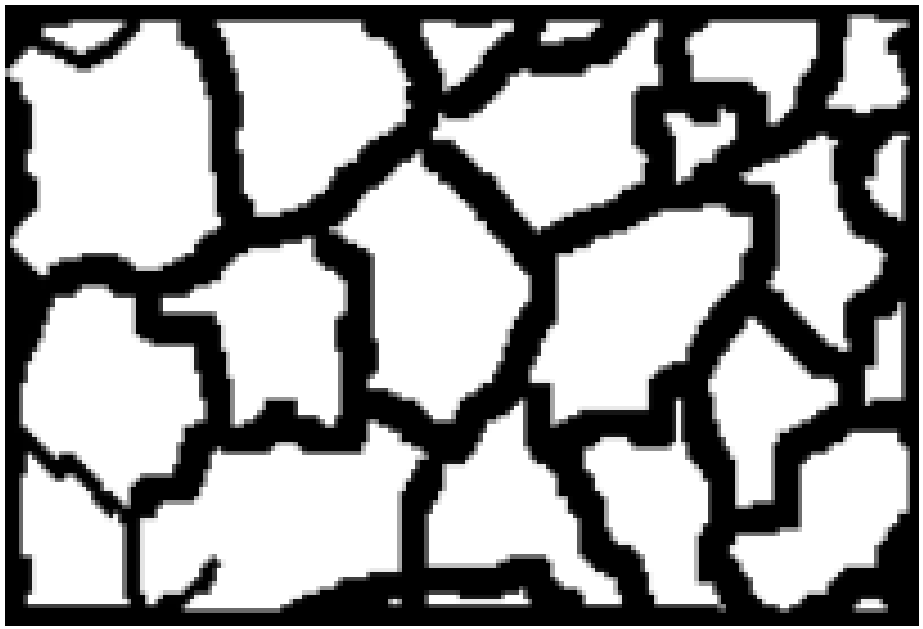
S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

Constraint Satisfaction Problem
Map Colouring

- Represent the map as a graph
  - Nodes are regions of the map
  - Edges between nodes indicate that two regions are adjacent
- Find an assignmens of colours to nodes such that no two adjacent nodes have the same colour

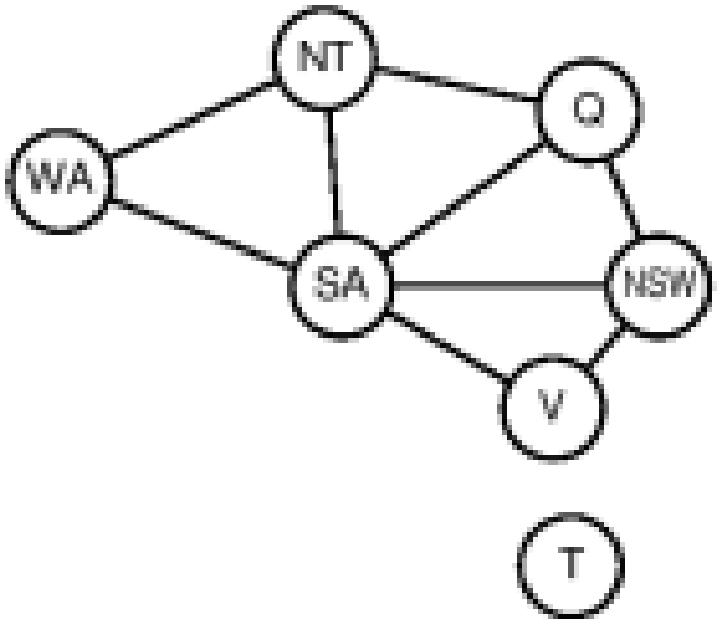
Map Colouring

- How many colours does it take to colour this map?
- Adjacent countries must have different colour
- Joined by a line (not a point)



Constraint Graph

- Nodes are variables
- Constraints are edges



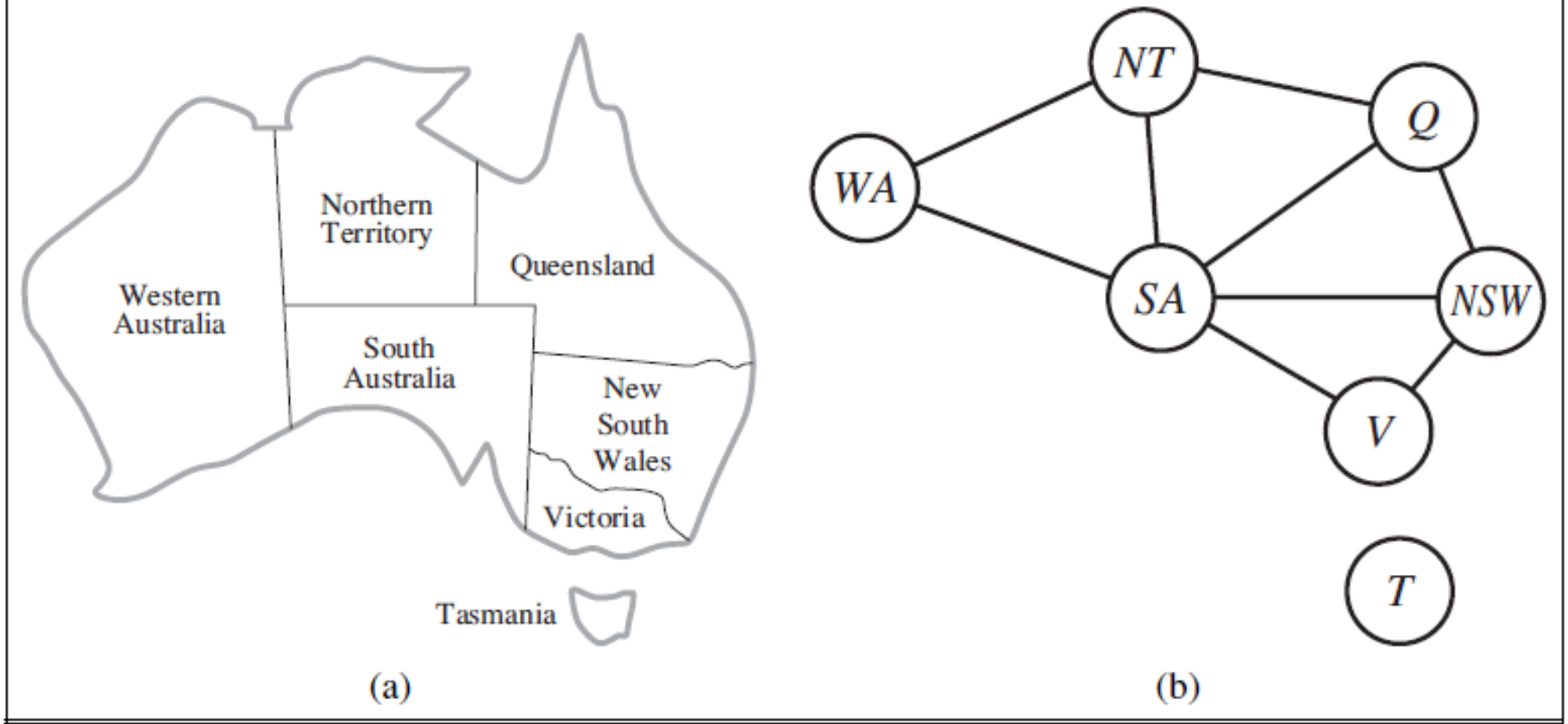


Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

INTRODUCTION TO GENETIC ALGORITHMS

- Simulating the logic of Darwinian selection, where only the best are selected for replication.
  - Over many generations, natural populations **evolve** according to the principles of natural selection and stated by Charles Darwin in The Origin of Species.
  - Only the **most suited elements** in a population are likely to survive and generate offspring, thus transmitting their biological heredity to new generations.
- Able to address complicated problems with many variables and a large number of possible outcomes by simulating the evolutionary process of **"survival of the fittest"** to reach a defined goal.
- Operate by **generating many random answers to a problem**, eliminating the worst and cross-pollinating better answers.
- **Repeating this elimination and regeneration process gradually improves the quality** of the answers to an optimal or near-optimal condition.

Five phases are considered in a genetic algorithm.

- **Initial population**

Set of individuals→Population→Characteristics→Gene→Chromosome

- **Fitness function**

Competing ability→fitness score→Higher score greater probability for selection

- **Selection**

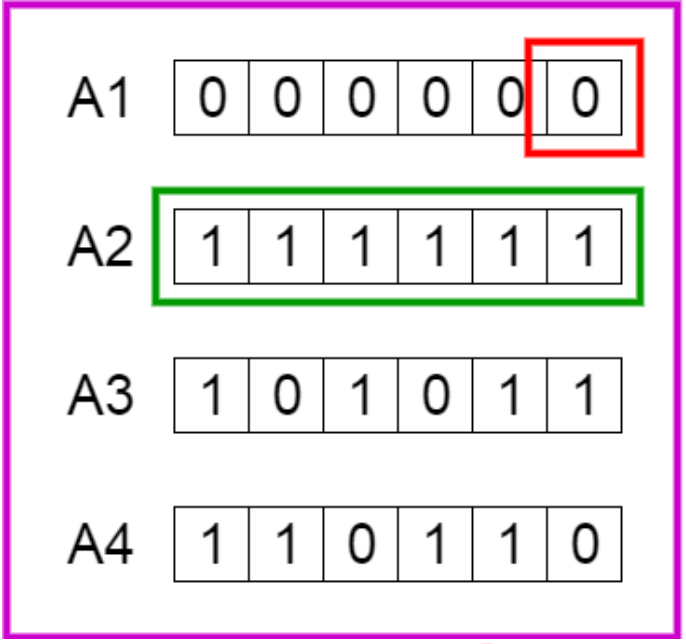
Fittest individuals selected to pass genes to the next generation.

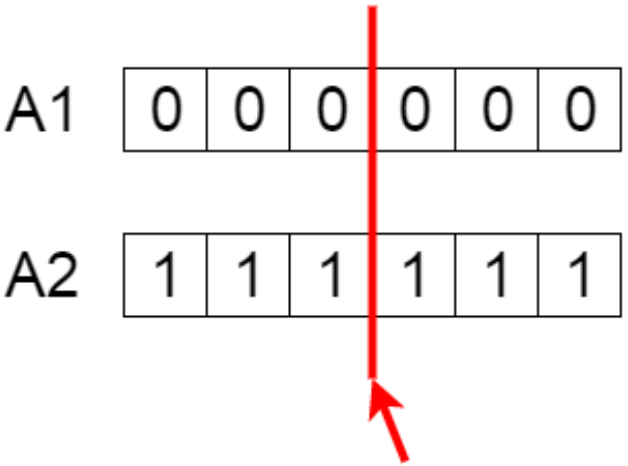
- **Crossover**

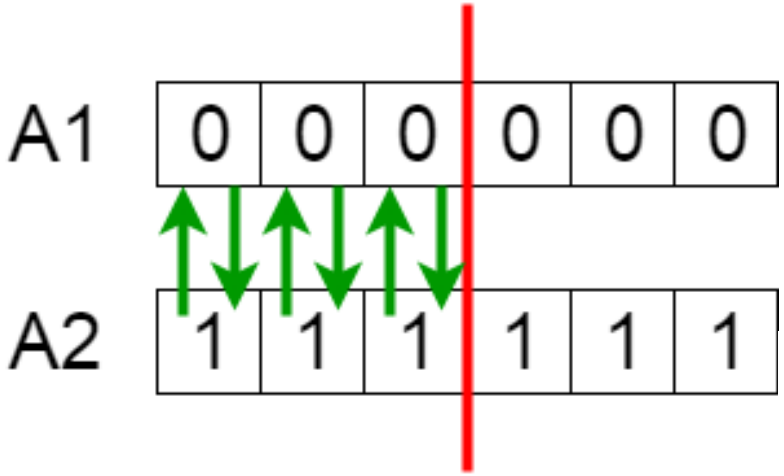
Randomly chosen Crossover point within the genes

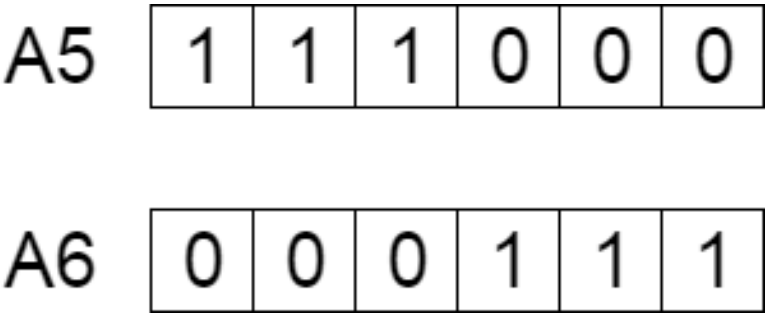
- **Mutation**

Some of the bits in the bit string can be flipped.

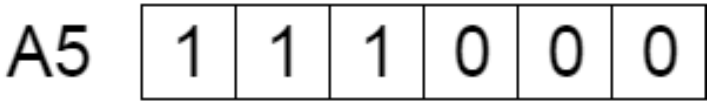




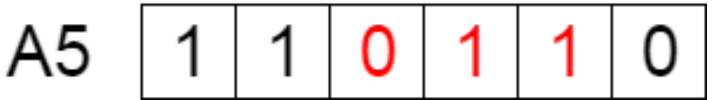




Before Mutation



After Mutation



Termination

- The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).
  - Then it is said that the genetic algorithm has provided a set of solutions to our problem.

START
Generate the initial population
Compute fitness
REPEAT
  Selection
  Crossover
  Mutation
  Compute fitness
UNTIL population has converged
STOP

GENETIC ALGORITHM

A variant of search in which successor states are generated by combining two parent states rather than by modifying a single state.

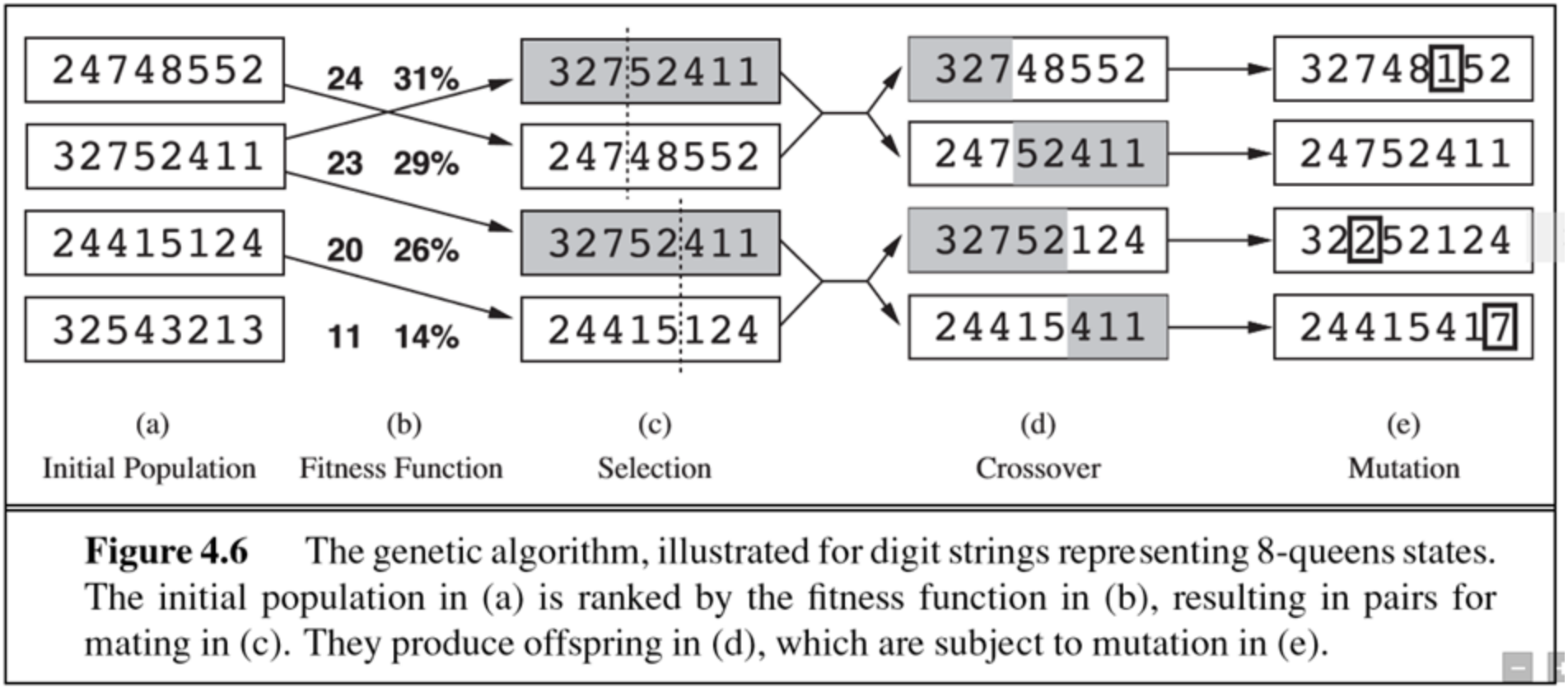
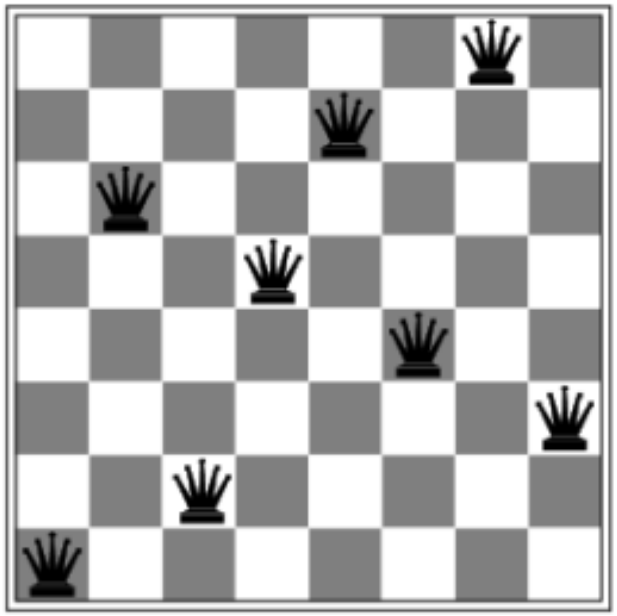


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

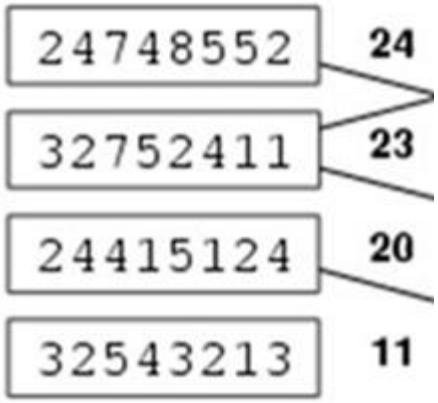
Example: 8 queens problem states

States: assume each queen has its own column, represent a state by listing a row where the queen is in each column (digits 1 to 8)

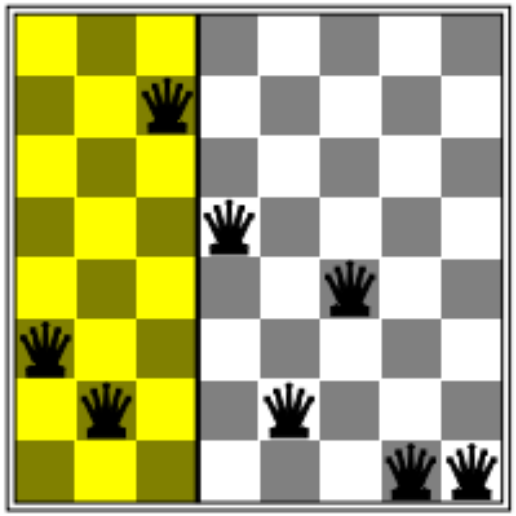
for example, the state below will be represented as 16257483



- **Starting from the queen in the left-most column just keep on counting the non-attacking positions (pairs) on the right with each queen. Carry on column by column towards your right until you reach the last queen. As a special case for the last queen the non-attacking pairs will be zero as their are no other queens after that.**

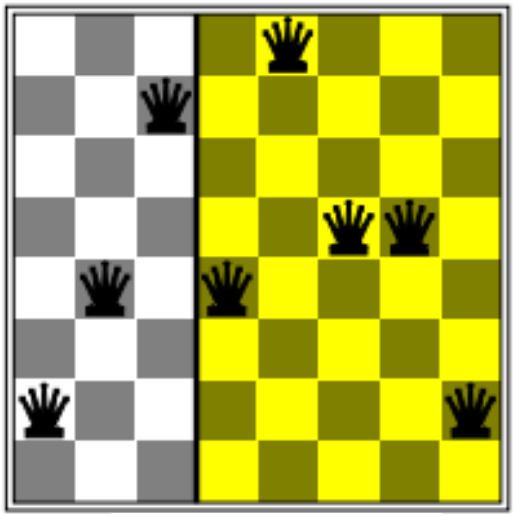


= 6 + 5 + 4 + 3 + 3 + 2



32752411

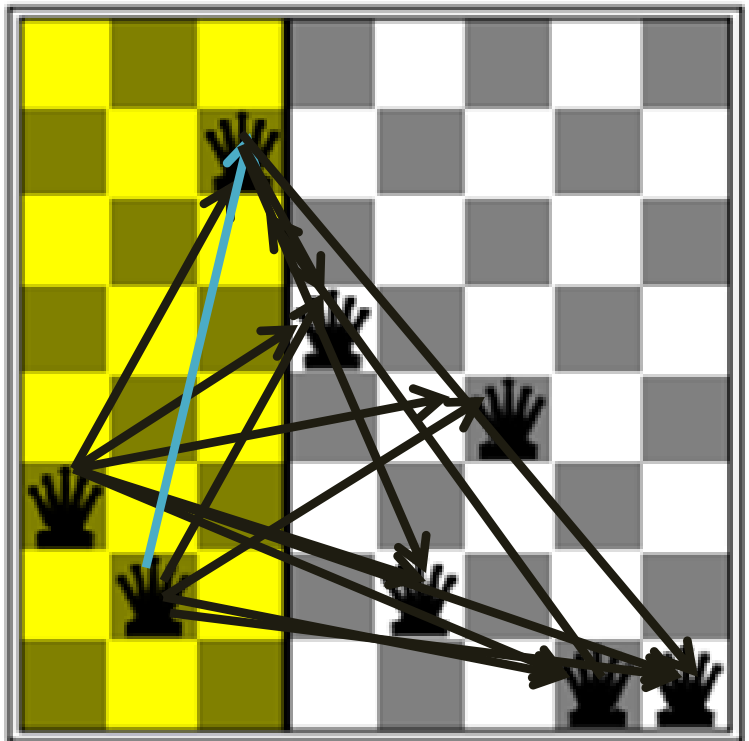
+



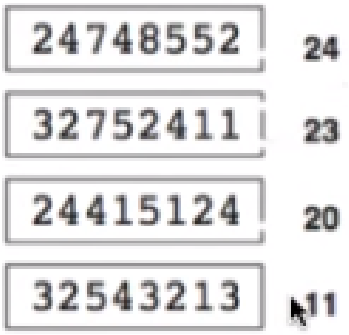
24748552

=

32748552

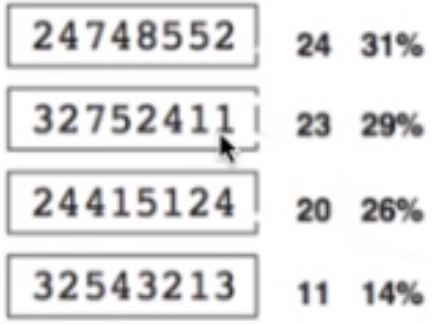


Represent states and compute fitness function.



(a)
Initial Population

Compute probability of being chosen (from fitness function).

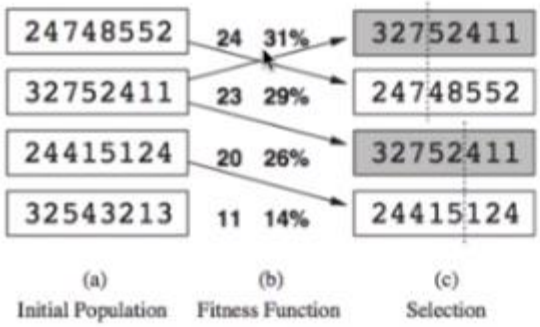


(a)
Initial Population

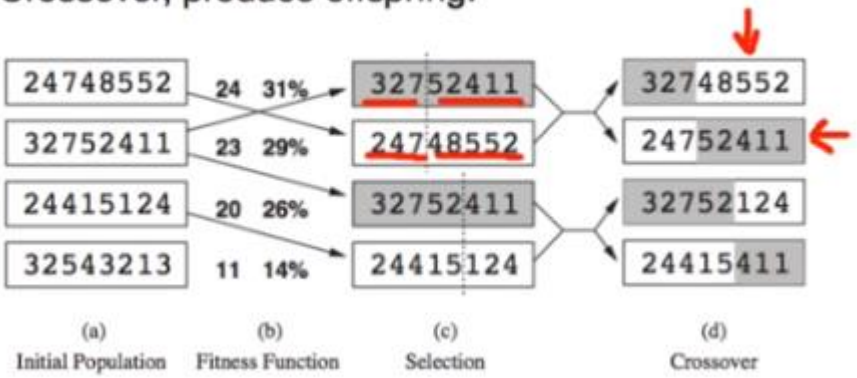
Genetic algorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc.

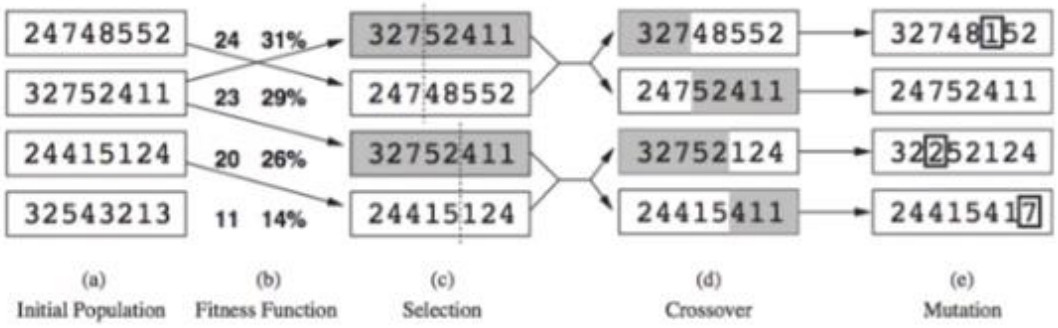
Randomly choose two pairs to reproduce based on probabilities. Pick a crossover point per pair.



Crossover, produce offspring.



May mutate.



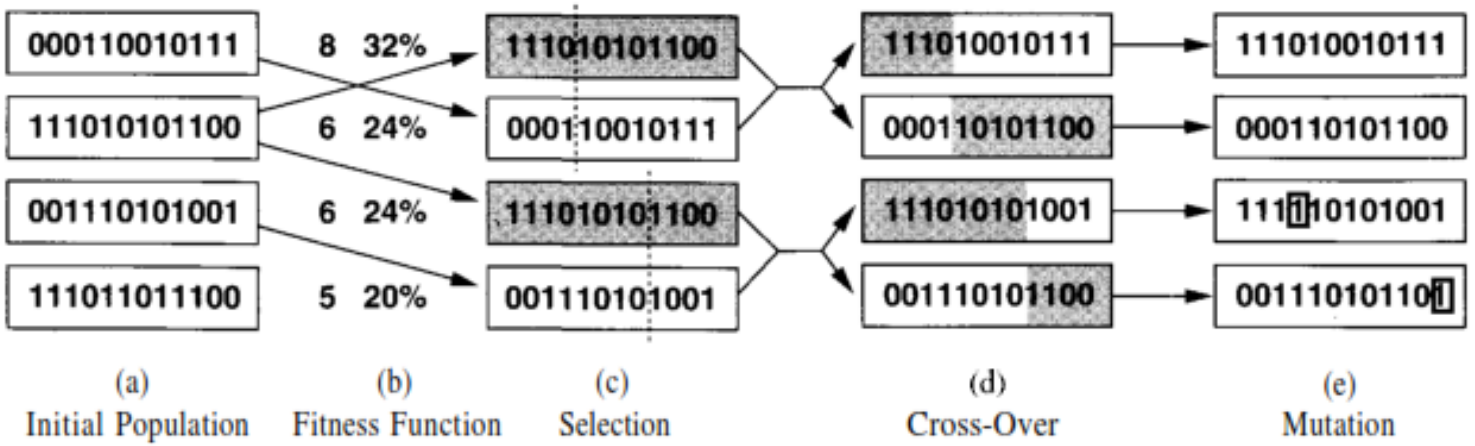


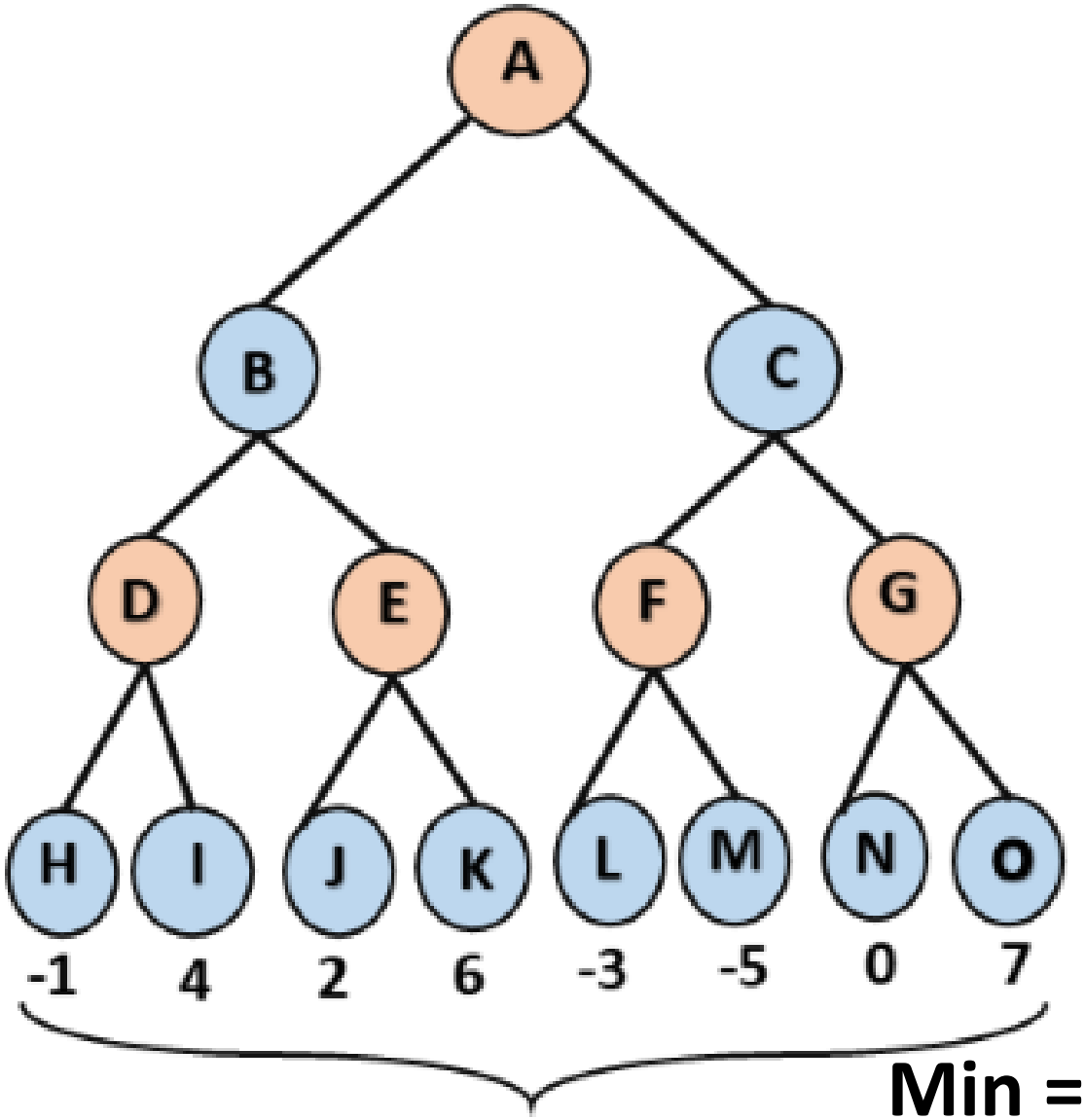
Figure 20.16 The genetic algorithm. In (a), we have an initial population of 4 individuals. They are scored by the fitness function in (b); the top individual scores an 8 and the bottom scores a 5. It works out that the top individual has a 32% chance of being chosen on each selection. In (c), selection has given us two pairs of mates, and the cross-over points (dotted lines) have been chosen. Notice that one individual mates twice; one not at all. In (d), we see the new offspring, generated by cross-over of their parents' genes. Finally, in (e), mutation has changed the two bits surrounded by boxes. This gives us the population for the next generation.

Like neural networks, genetic algorithms are easy to apply to a wide range of problems. The results can be very good on some problems, and rather poor on others. In fact, Denker's remark that "neural networks are the second best way of doing just about anything" has been extended

ADVERSARIAL SEARCH/GAMES

- A **search when there is an "enemy" or "opponent" changing the state of the problem every step in a direction you "DO NOT" want.**
- Examples: Chess, business, trading, war.
  - You change state, but then you don't control the next state.
  - Opponent will change the next state in a way: unpredictable.
- Min Max algorithm
- Alpha beta pruning

- Adversarial Search for the minimax procedure works as follows:
  - It aims to find the optimal strategy for MAX to win the game.
  - It follows the approach of Depth-first search.
  - In the game tree, optimal leaf node could appear at any depth of the tree.
  - Propagate the minimax values up to the tree until the terminal node discovered.



Maximizer

Minimizer

Maximizer

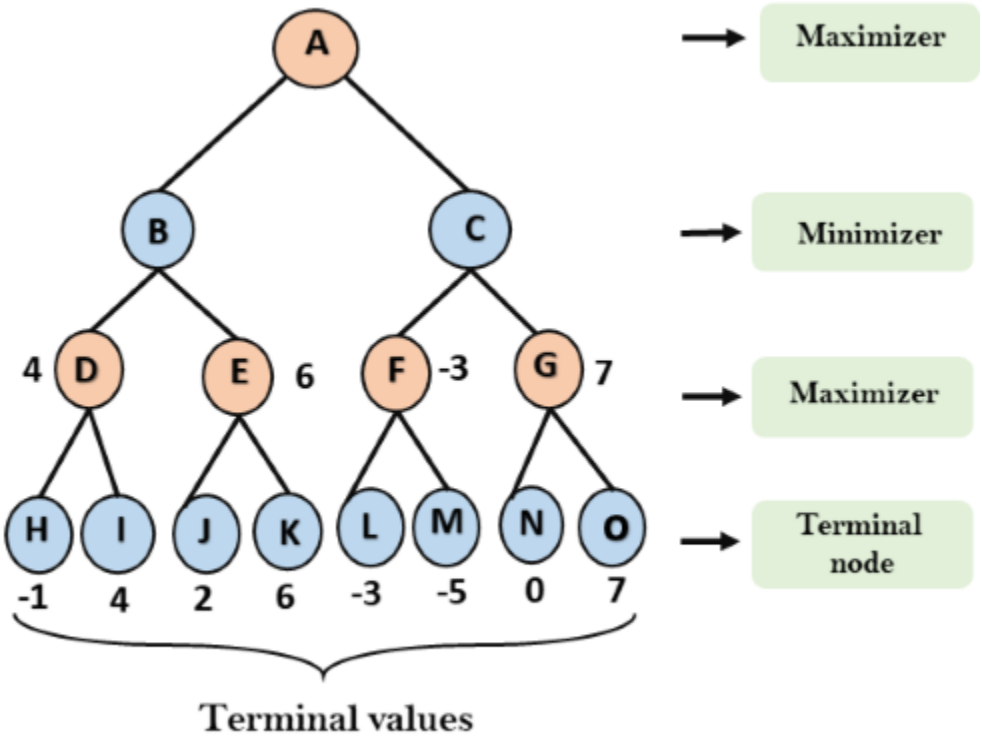
Terminal node

Min = + ∞

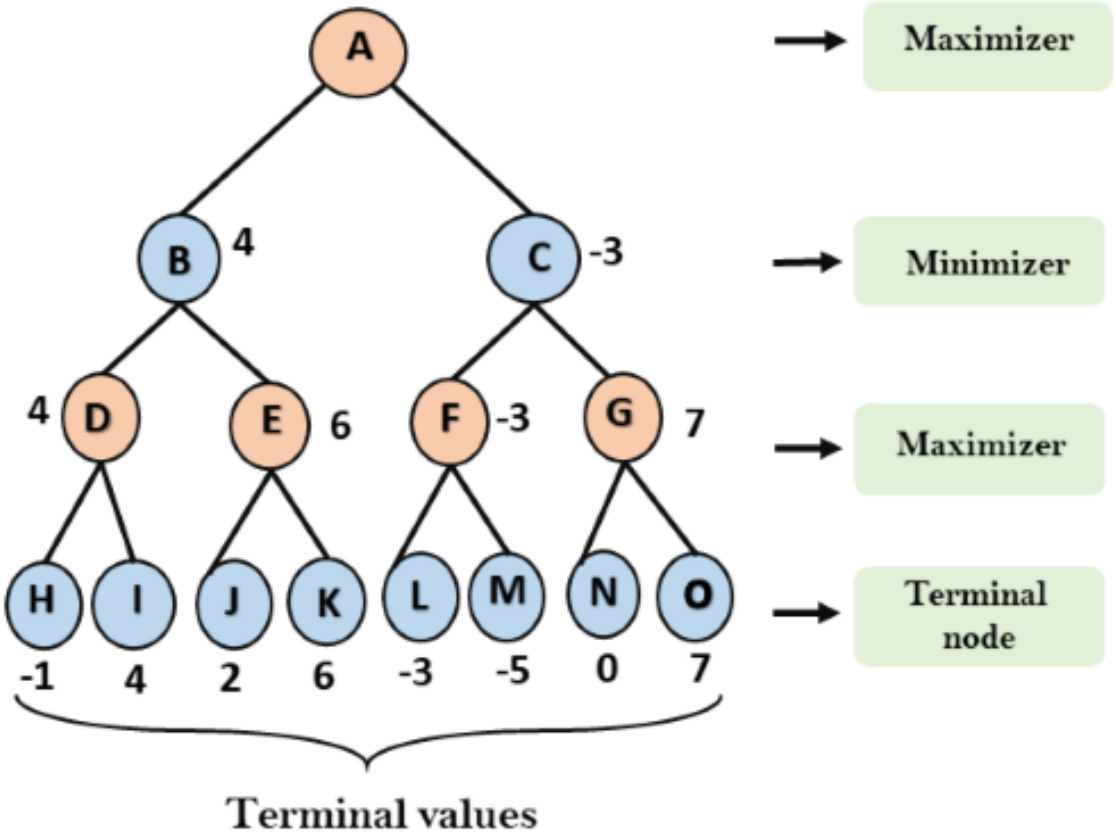
worst-case initial value = + infinity.

Max = - ∞

worst-case initial value = - infinity

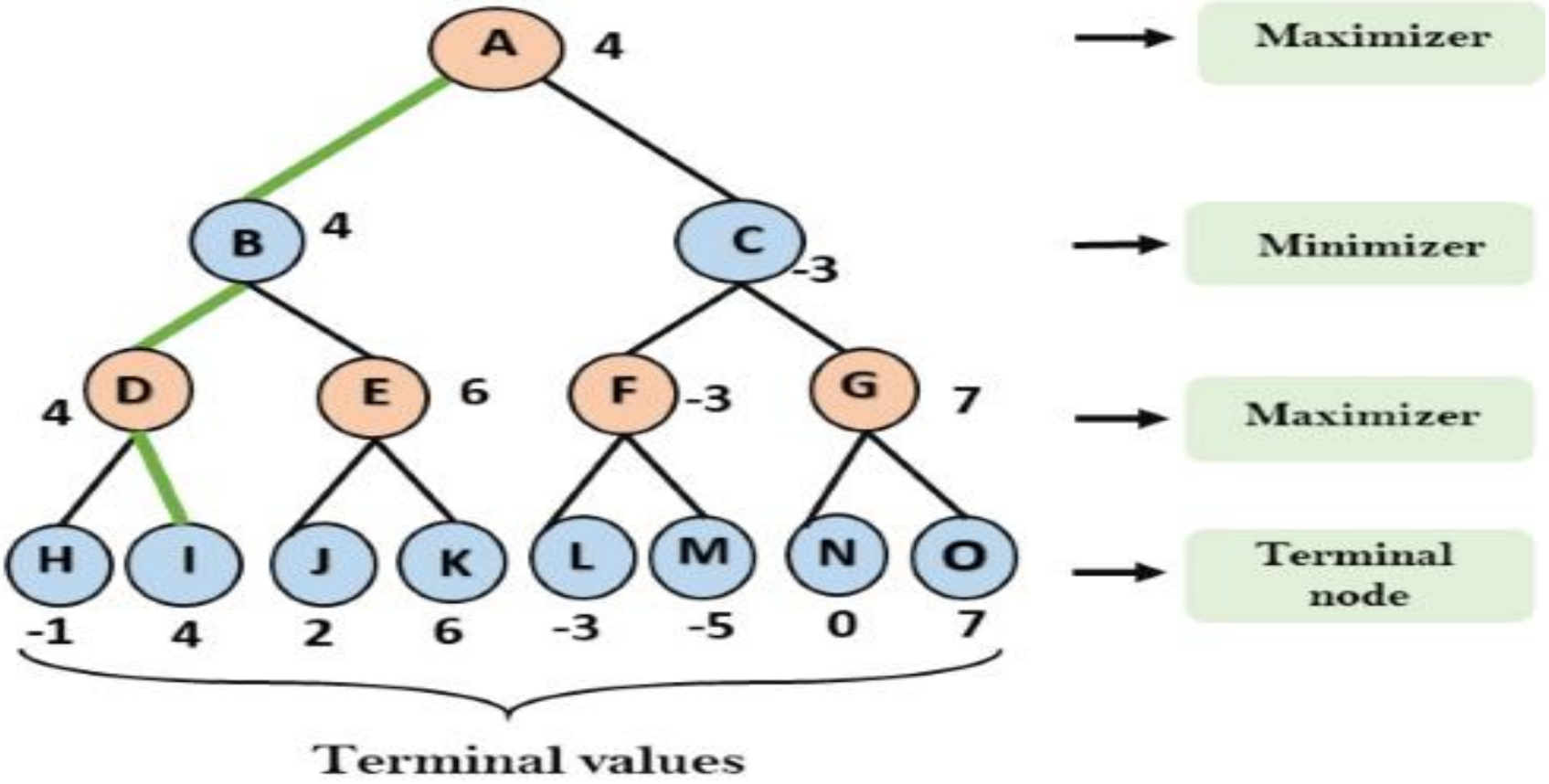


- For node D max(-1,- -∞) => max(-1,4) = 4
- For Node E max(2, -∞) => max(2, 6) = 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



For node B= min(4,6) = 4

For node C= min (-3, 7) = -3



For node A max(4, -3)= 4

Alpha-beta pruning

A method that optimizes the Minimax algorithm

This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**./**Alpha-Beta Algorithm**.

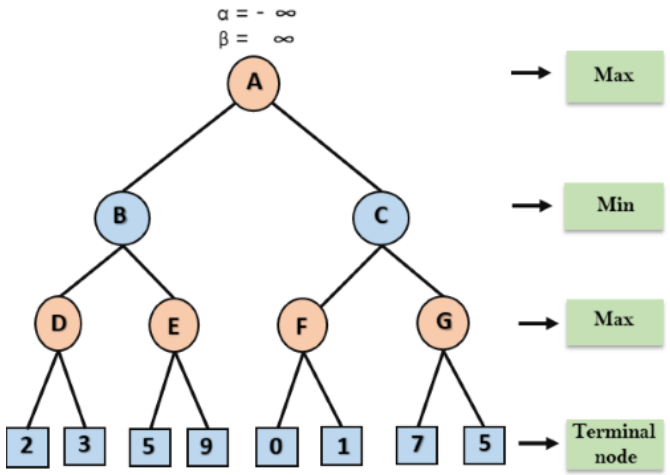
- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

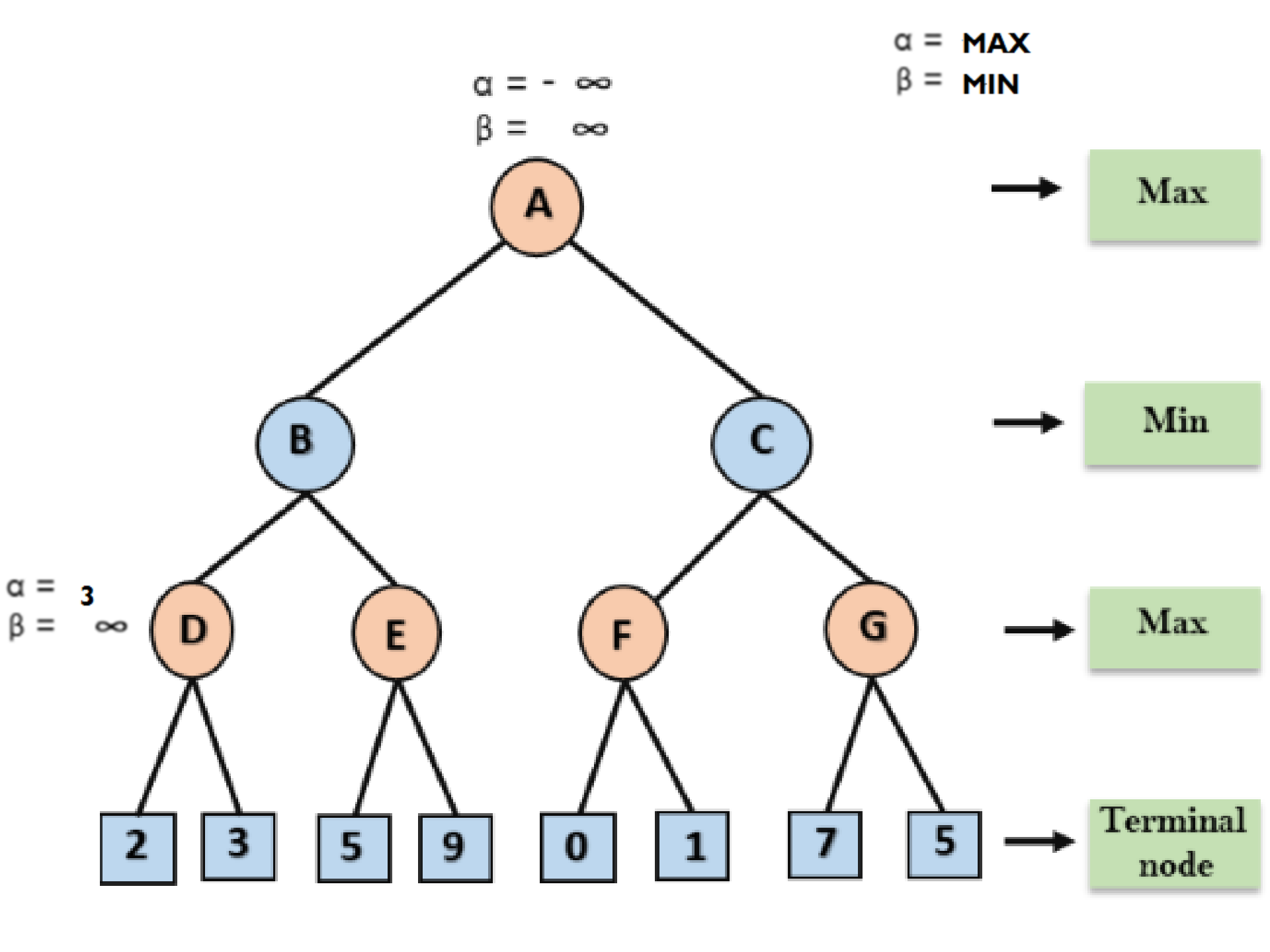
Condition for Alpha-beta pruning: α>=β

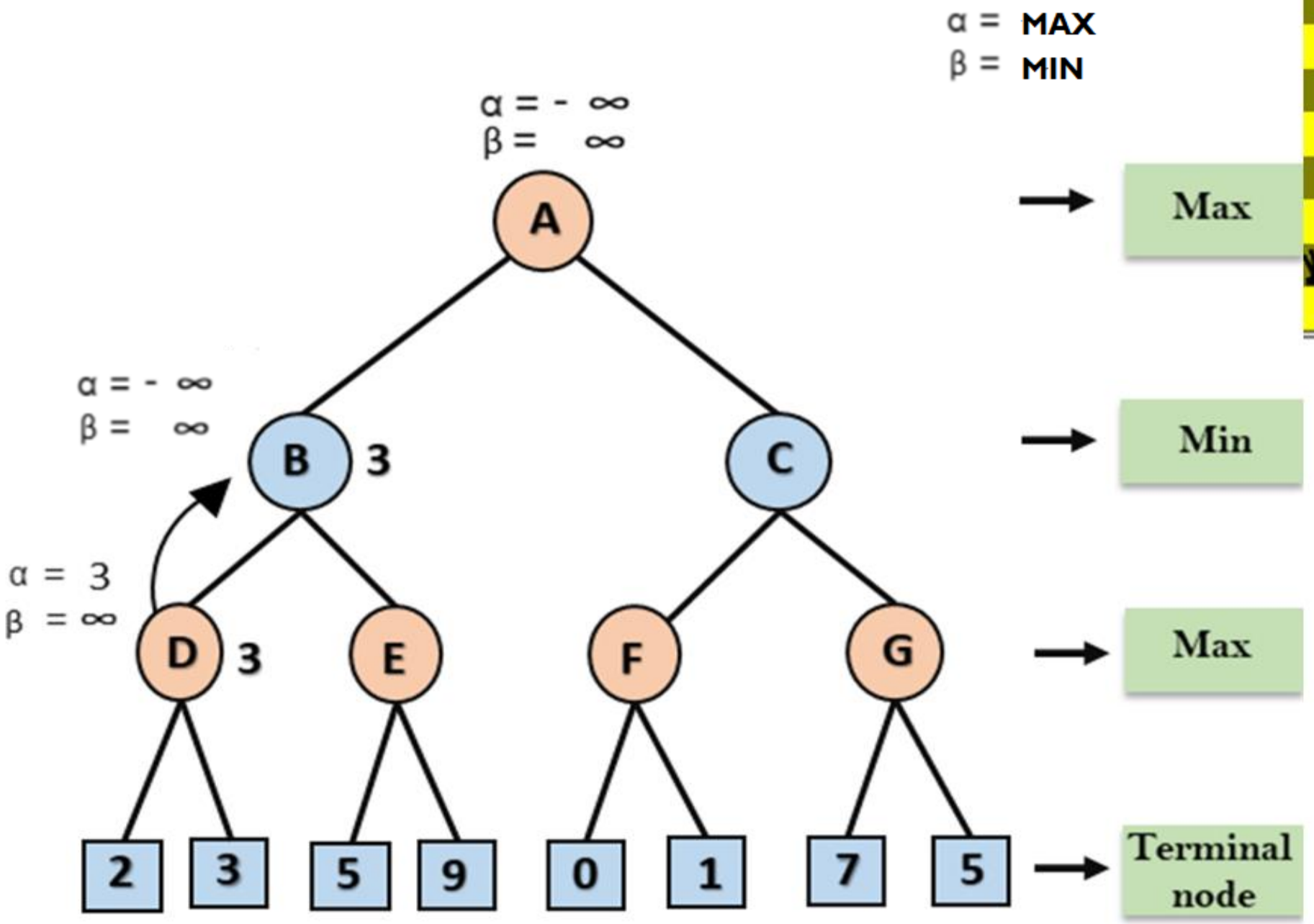
- The **Max player** will only update the value of **alpha**.
- The **Min player** will only update the value of **beta.**

- We will only pass the alpha, beta values to the child nodes.
  - No values will be sent upwards the tree

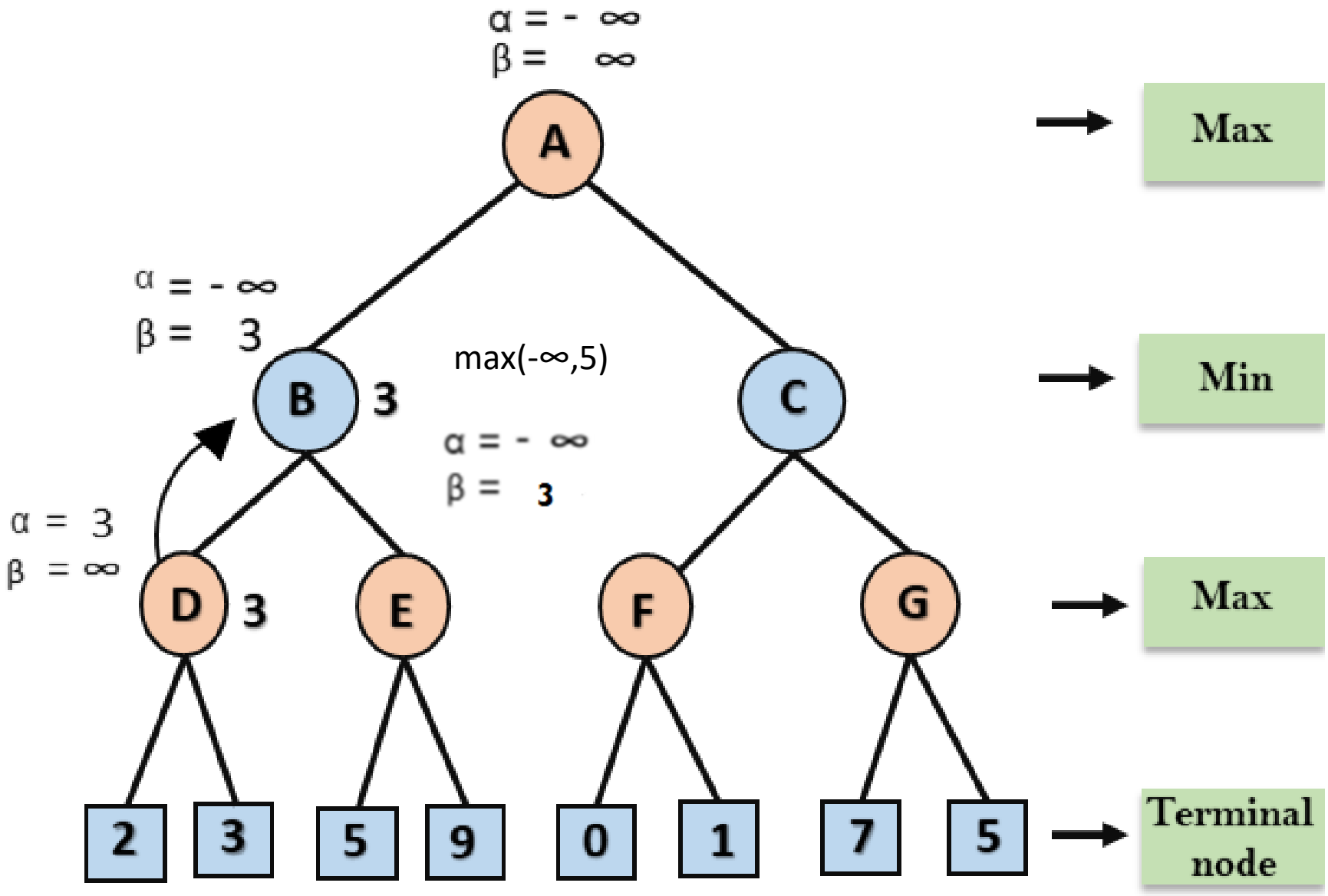
Depth First Approach with Backtracking.

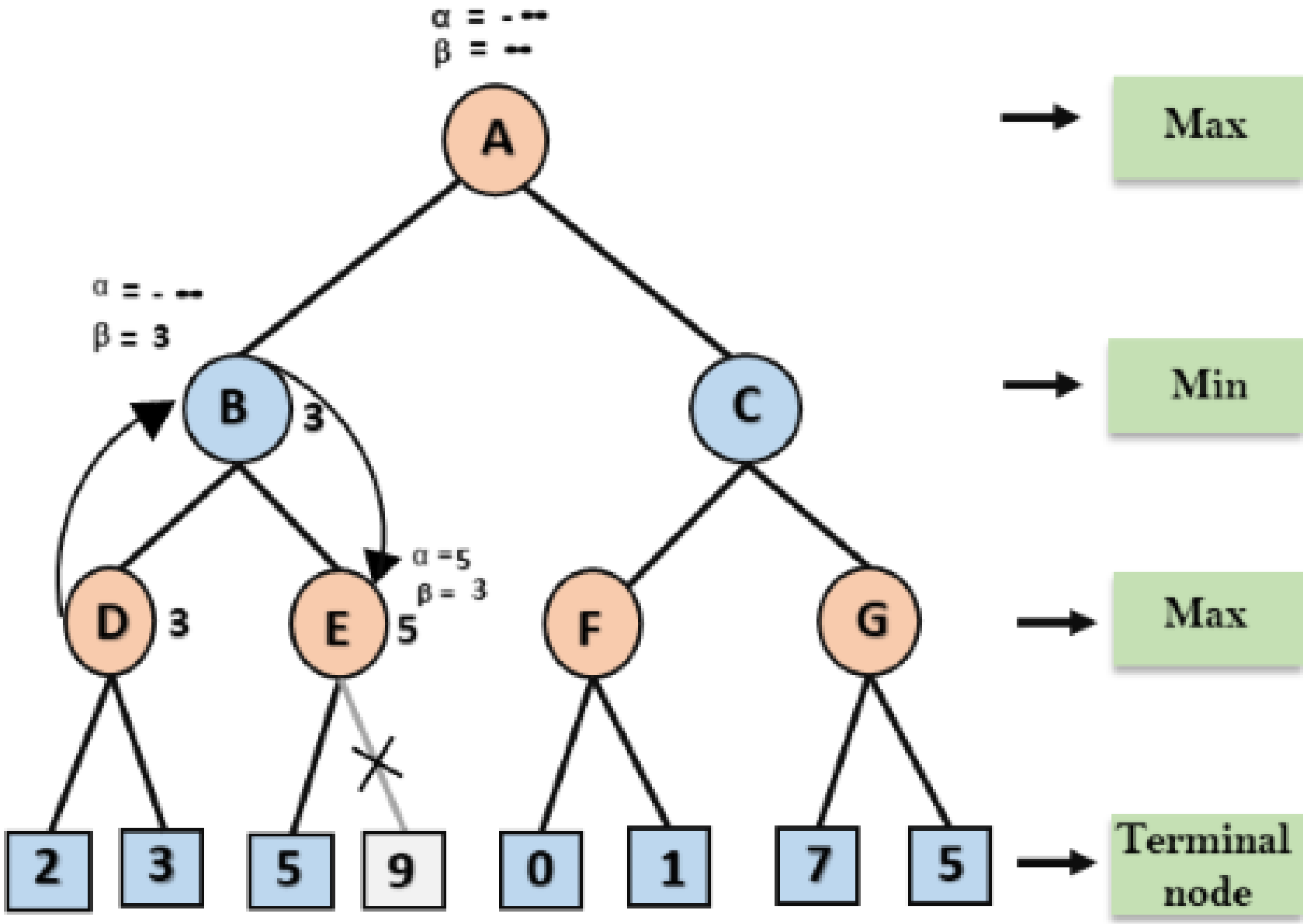


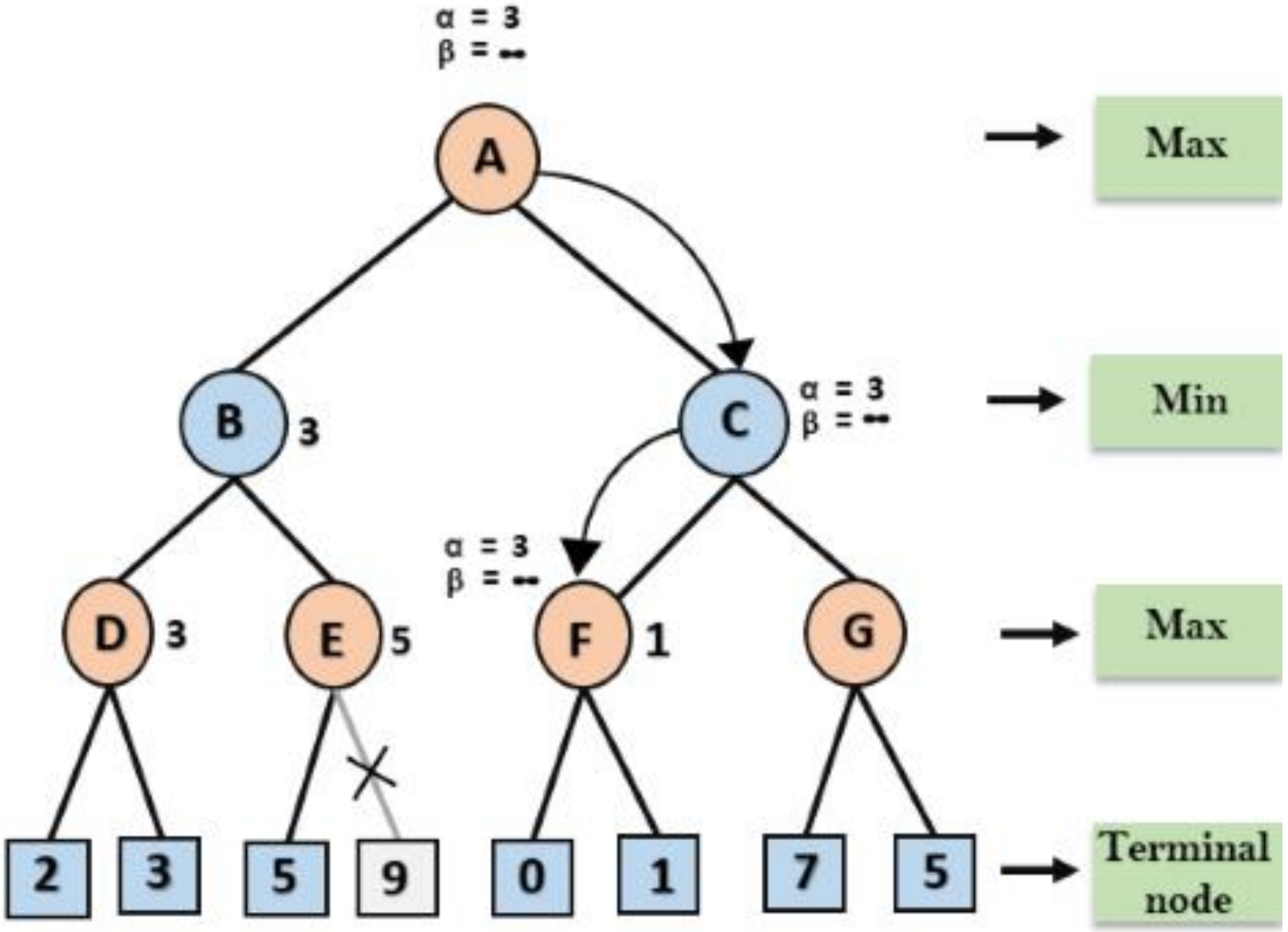

α = MAX
β = MIN
α = - ∞
β = ∞
A
B
C
D
E
F
G
2
3
5
9
0
1
7
5
α = 3
β = ∞
Max
Min
Max
Terminal node

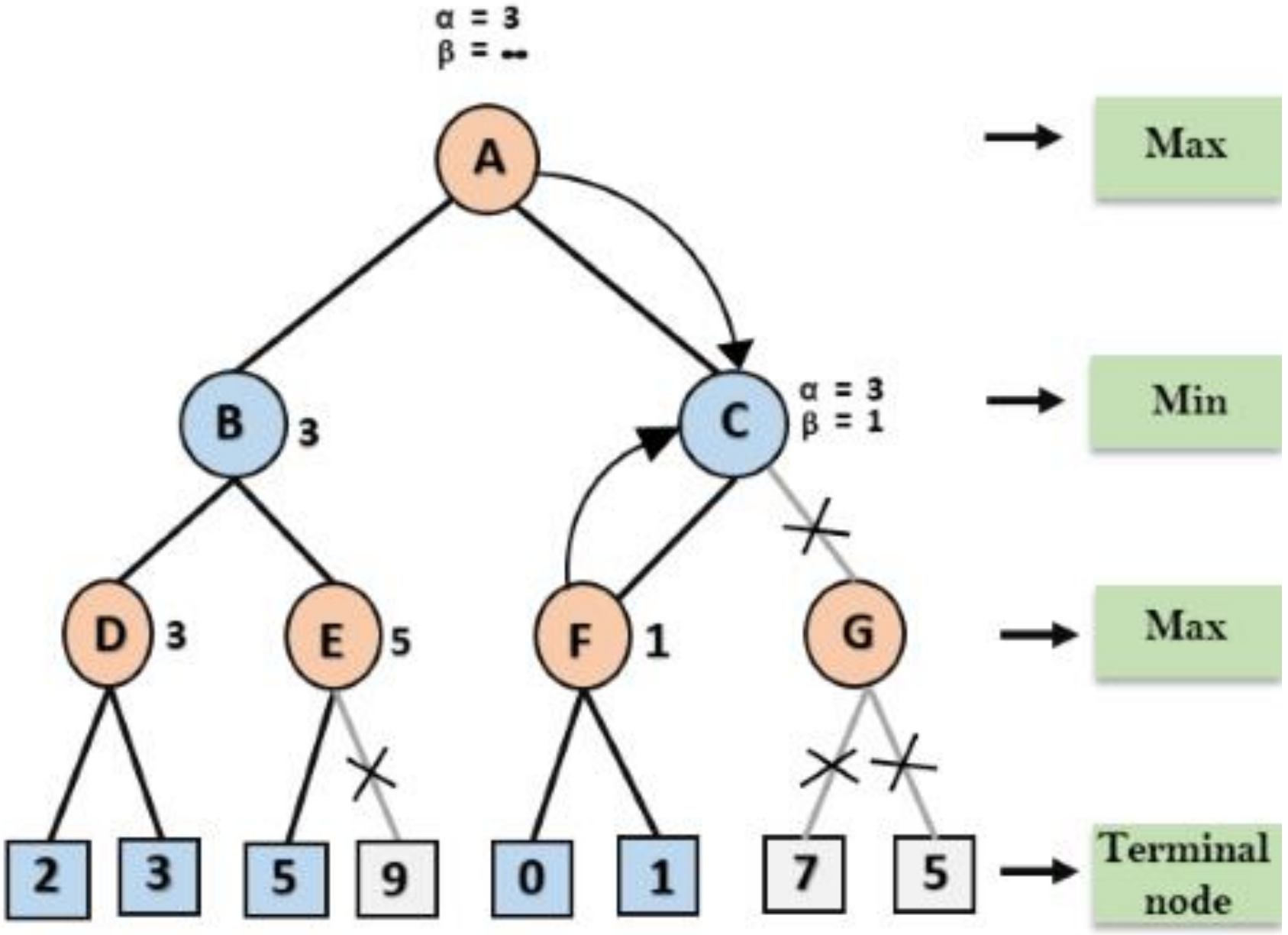

α = MAX
β = MIN
α = - ∞
β = ∞
A
Max
α = - ∞
β = ∞
B
3
C
Min
α = 3
β = ∞
D
3
E
F
G
Max
2
3
5
9
0
1
7
5
Terminal node

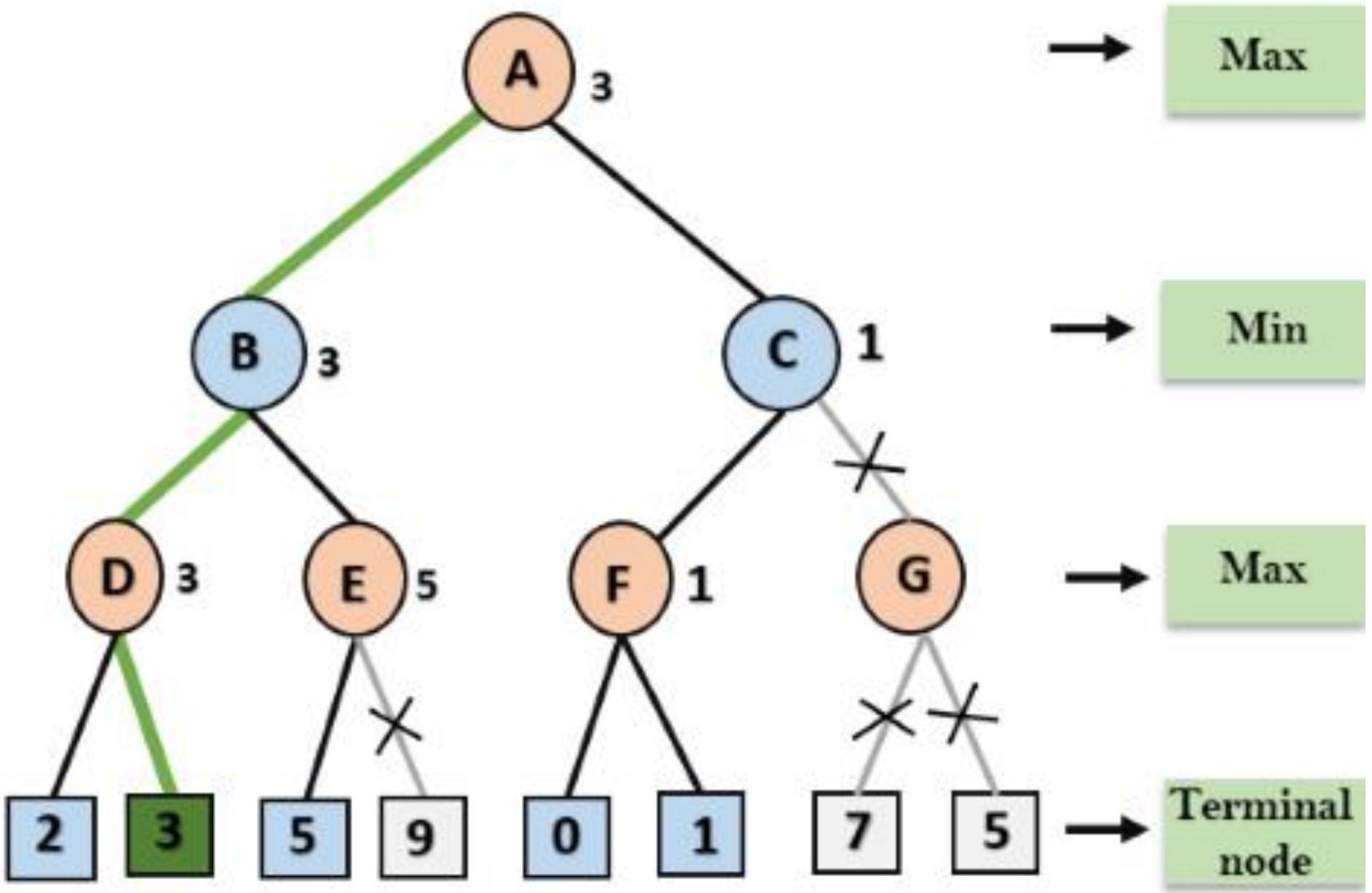
α = MAX
β = MIN


α = - ∞
β = ∞
A
Max
α = - ∞
β = 3
B
3
max(-∞,5)
α = - ∞
β = 3
C
Min
α = 3
β = ∞
D
3
E
F
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = -∞
β = ∞
A
Max
α = -∞
β = 3
B
3
C
Min
α =5
β = 3
D
3
E
5
F
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = 3
β = ∞
A
Max
B
3
C
α = 3
β = ∞
Min
α = 3
β = ∞
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = 3
β = ∞
A
Max
B
3
C
α = 3
β = 1
Min
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node


A
3
B
3
C
1
D
3
E
5
F
1
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node

Minimax with alpha-beta pruning on a two-person game tree of 4 plies