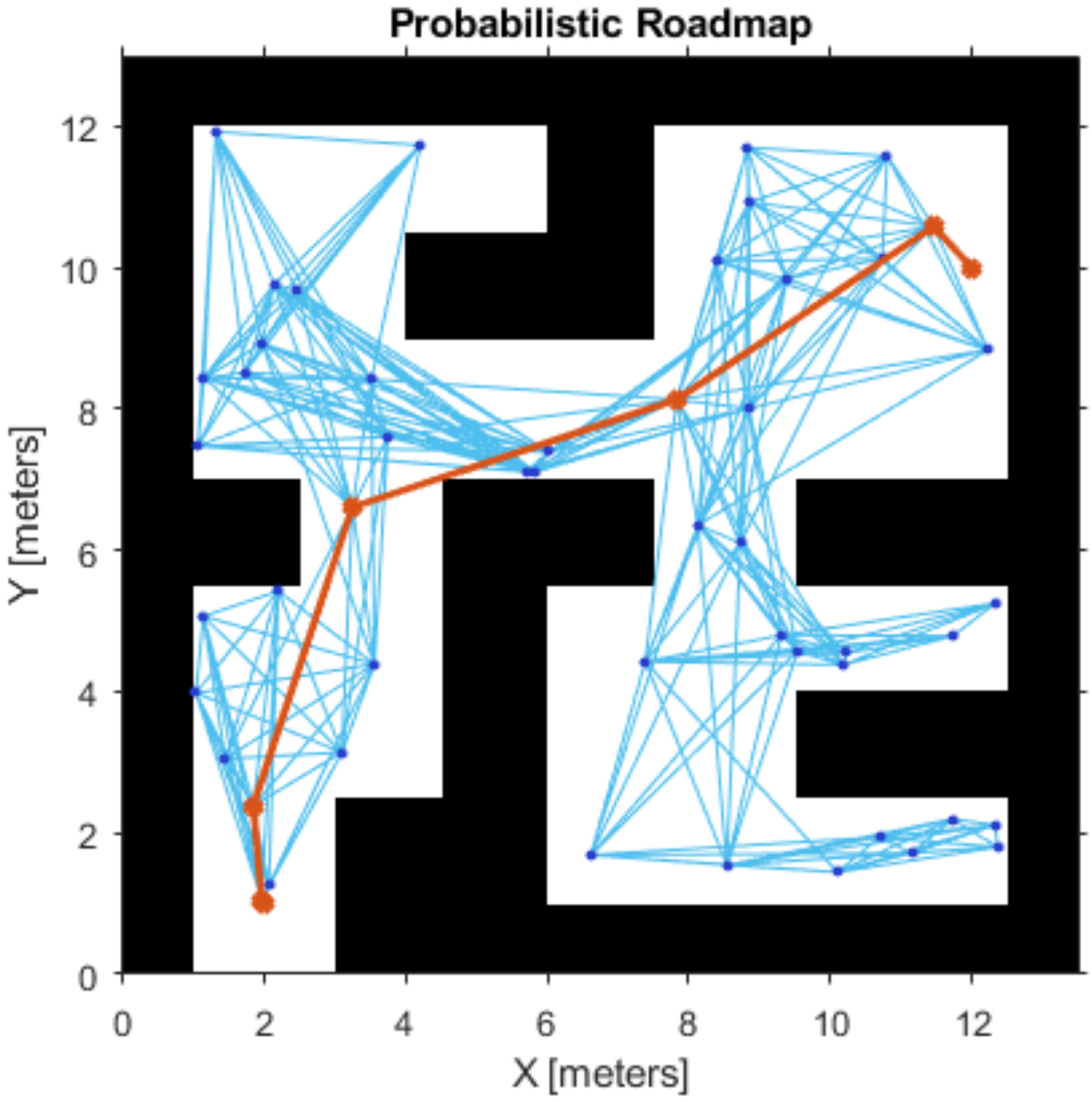
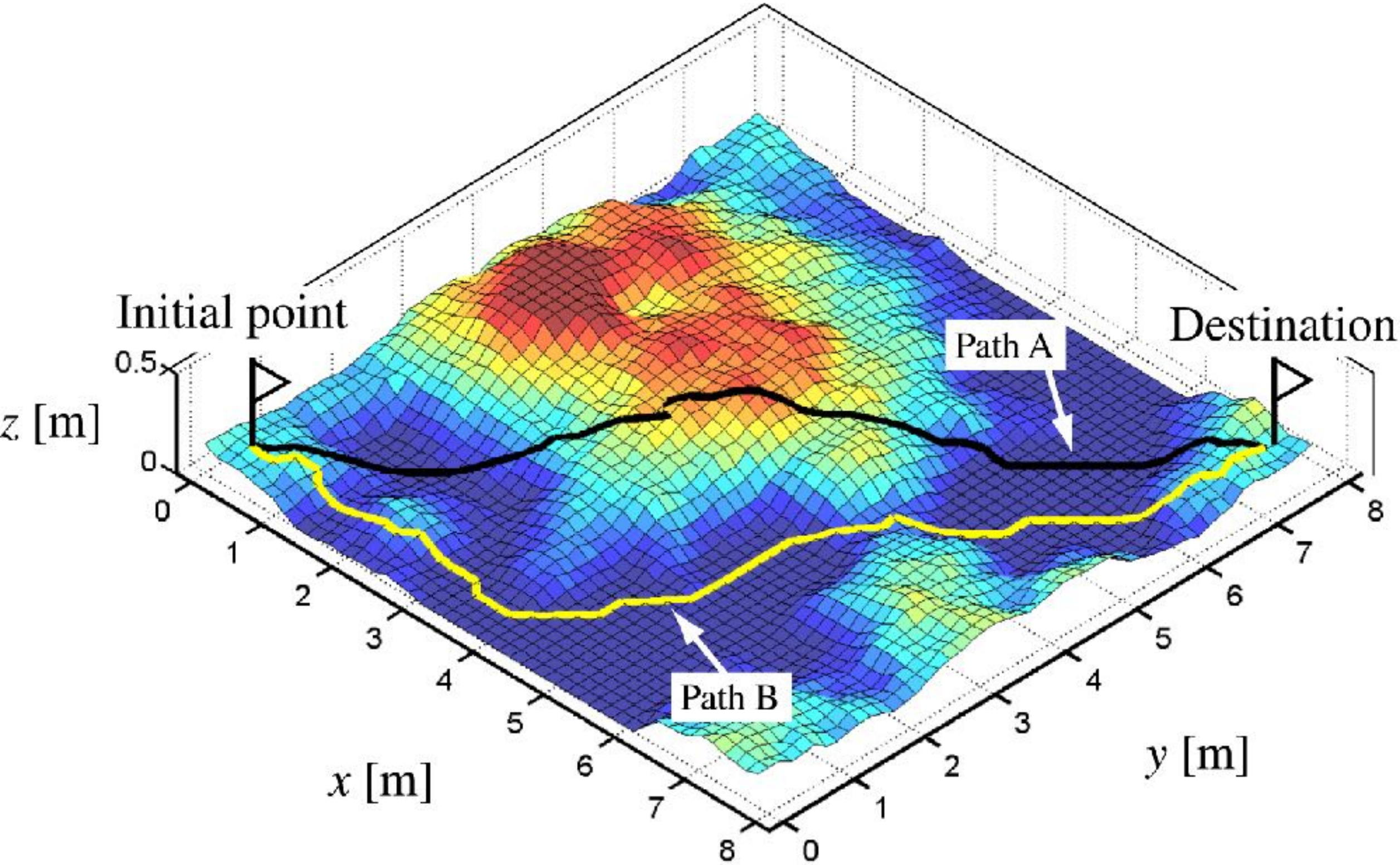
ARTIFICIAL INTELLIGENCE

Problem Solving

PATH PLANNING



Retrieved from: https://www.mathworks.com/help/examples/robotics_product/win64/PathPlanningExample_03.png



Retrieved from: http://www.astro.mech.tohoku.ac.jp/~ishigami/research/image/path.jpg

STATE SPACE

BEFORE SOLVING THE PROBLEM

You are given a problem to solve.

What do you need to do first?

- For example, when you want to complete a jigsaw puzzle, what do you do first?

BEFORE SOLVING THE PROBLEM

What do you need to do first?

Retrieved from https://lh6.ggpht.com/v_0cgItjMUfOzijN1Pq5pIteMBjmo7NXhUmx8kzkGD1irQ-dH2-_s-Y7hhes3q-KjxJz=h900

BREAKING DOWN THE PROBLEM

Identify:

- Initial State
    - What does the problem look like in the beginning?
- Current State
    - What does the problem look like right now?
- Goal State
    - What is the final goal?
- Actions we can take
    - What can we act on the problem?
- Result after the action is taken
    - What happens after we perform the action.

STATE SPACE

S = <S, A, Action(s), Result(s,a), Cost(s,a)>

S = Set of all states

A = Set of all action

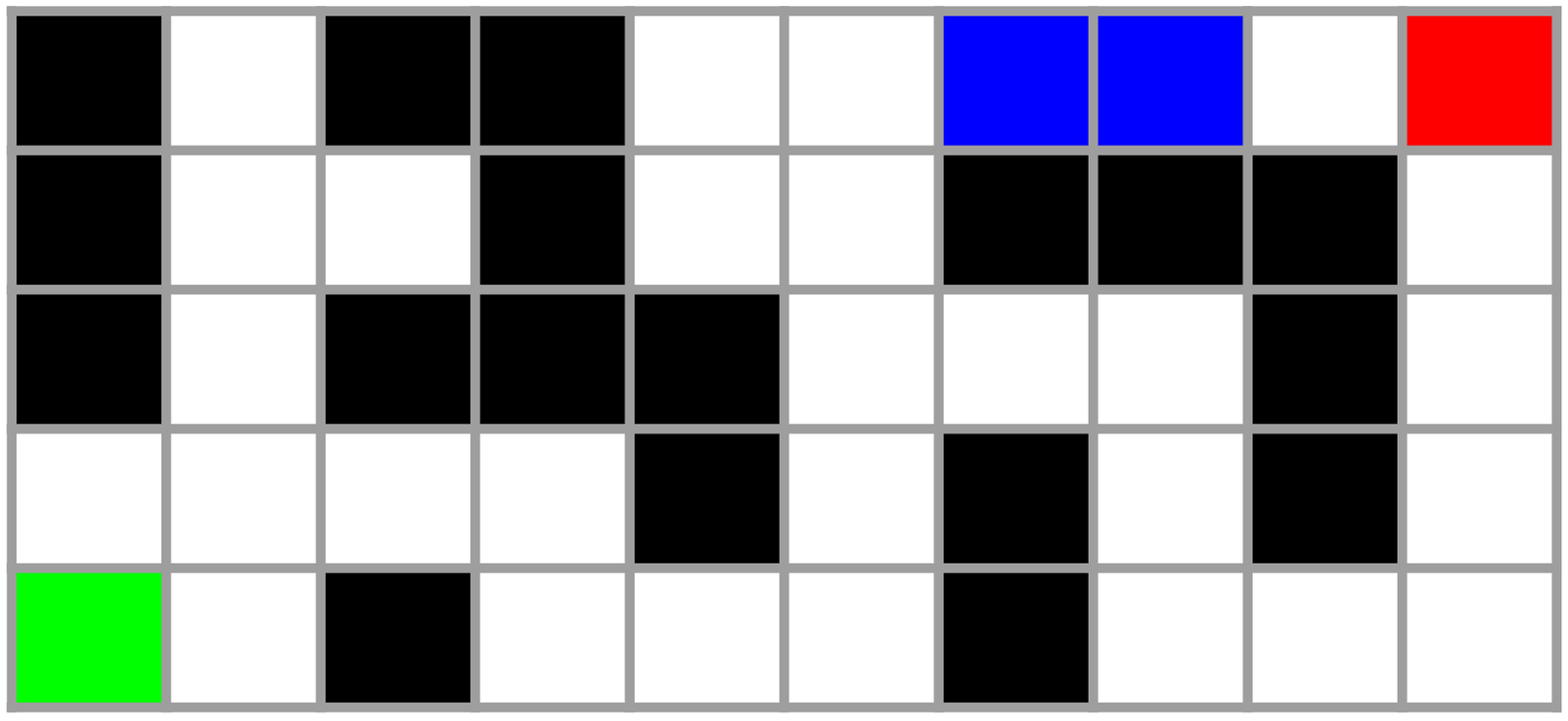
Action(s) = All valid actions at state s

Result(s,a) = State after taking action a at state s

Cost(s,a) = Cost associated when taking action a at state s

PROBLEM FORMATION – ROBOT NAVIGATION

- Given that a robot starts at the green square and tries to go to the red square. The robot can move up, down, left, and right. Black squares represents impassable terrain. White squares represent passable terrain. And blue squares represent hard-to-navigate terrain. How do we set up this state space?



SEARCHING FOR SOLUTION

SEARCHING

We have

- State Spaces
- Initial State
- Goal State

Our job is to find a way to move from Initial State to Goal State

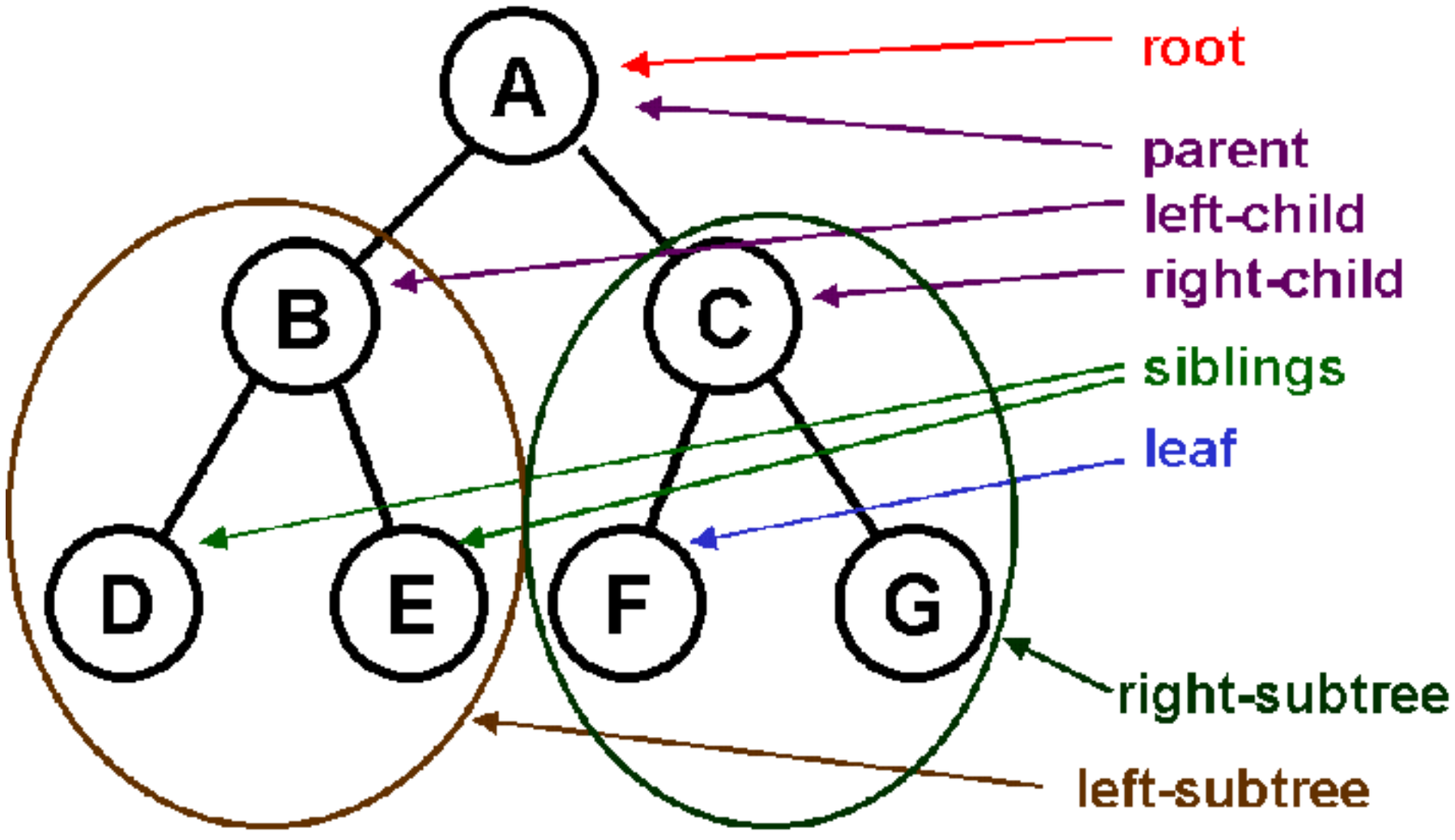
We also need to find a way to represent these actions

- Tree
- Graph
- Grid

ADVANCED DATA STRUCTURE

TREE

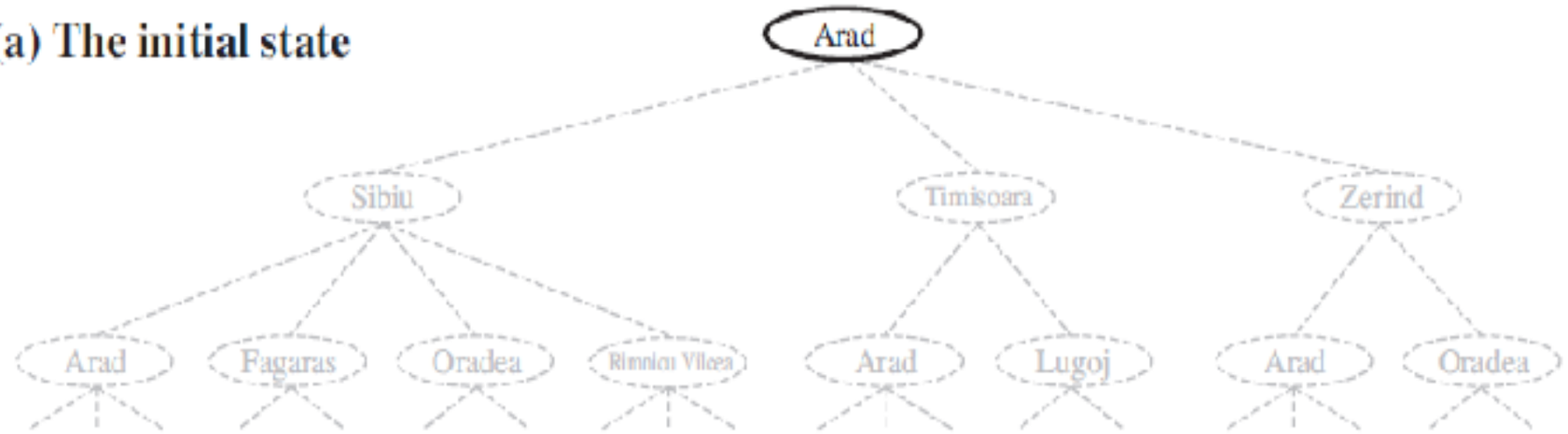
- Nodes with multiple links to other nodes.



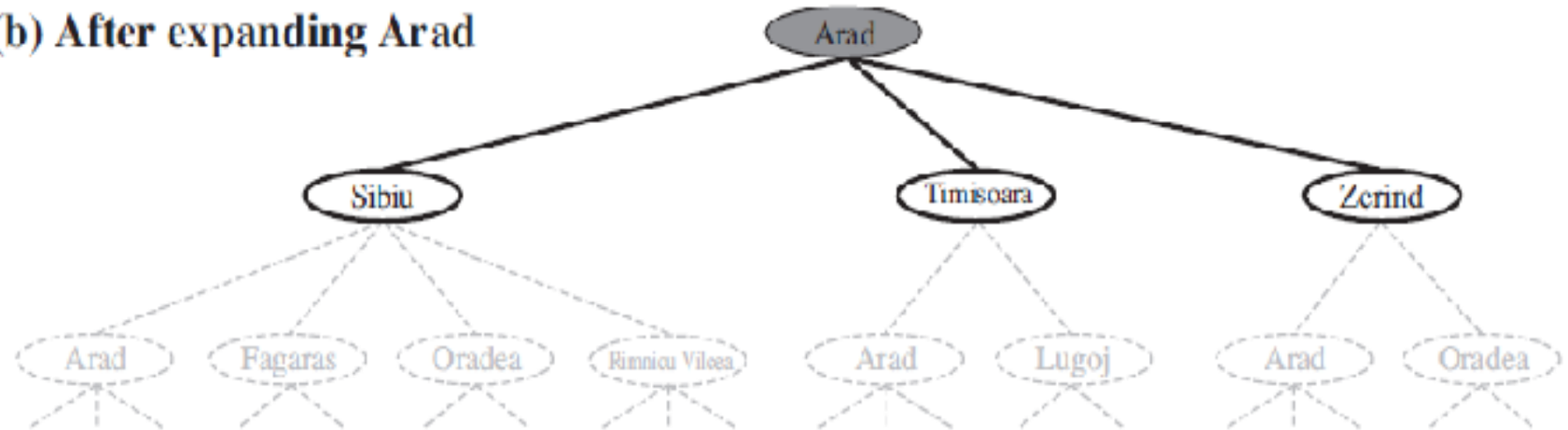
Retrieved from: http://2.bp.blogspot.com/-CApCcTOe1A0/TwppaUWiQsI/AAAAAAAABX4/sJv92_ZJzhE/s1600/Tree+Term.gif

TREE

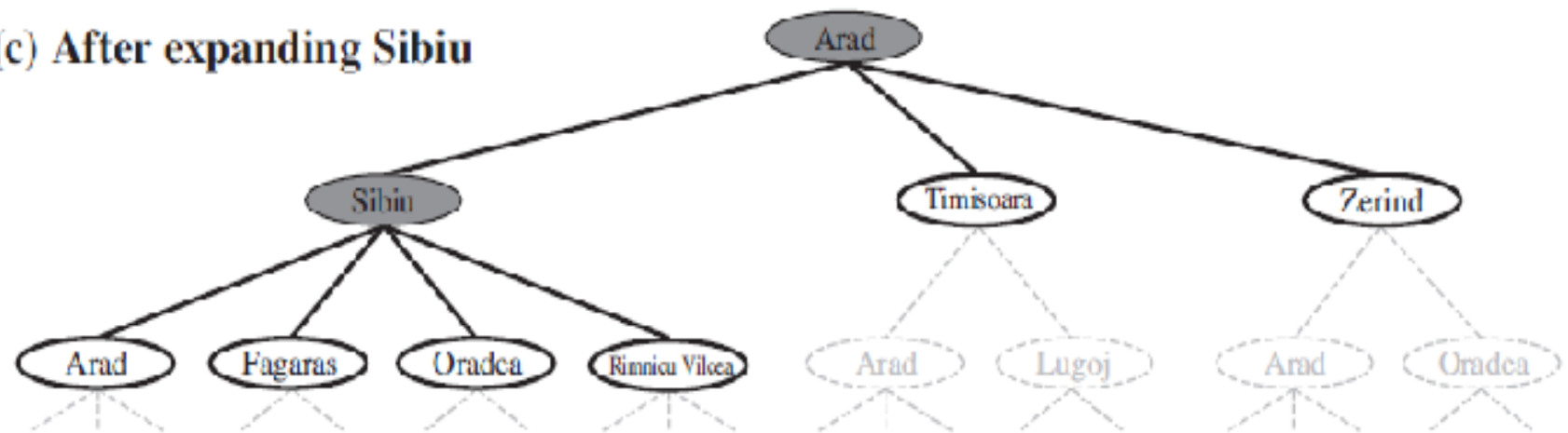
(a) The initial state



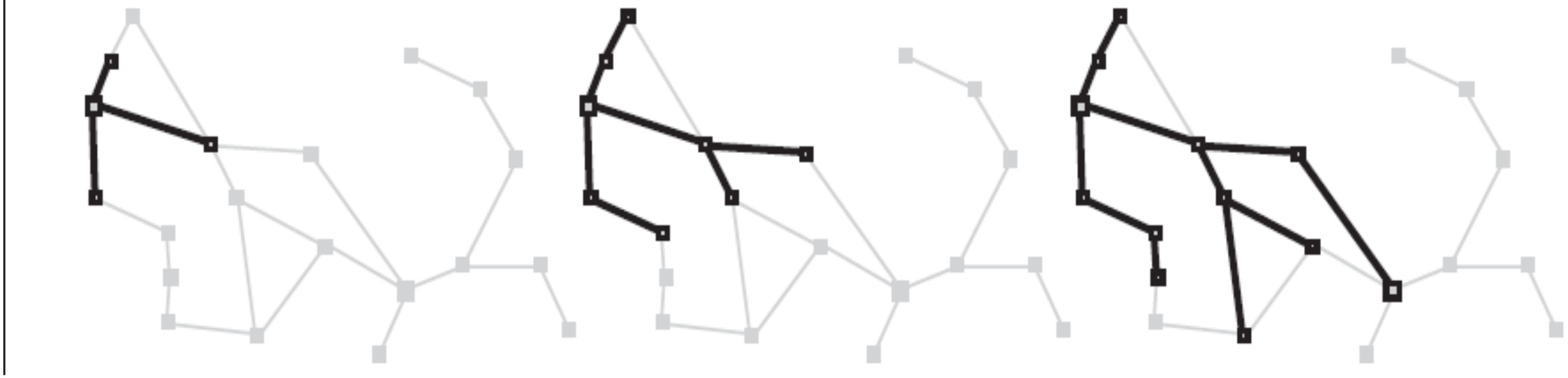
(b) After expanding Arad



(c) After expanding Sibiu

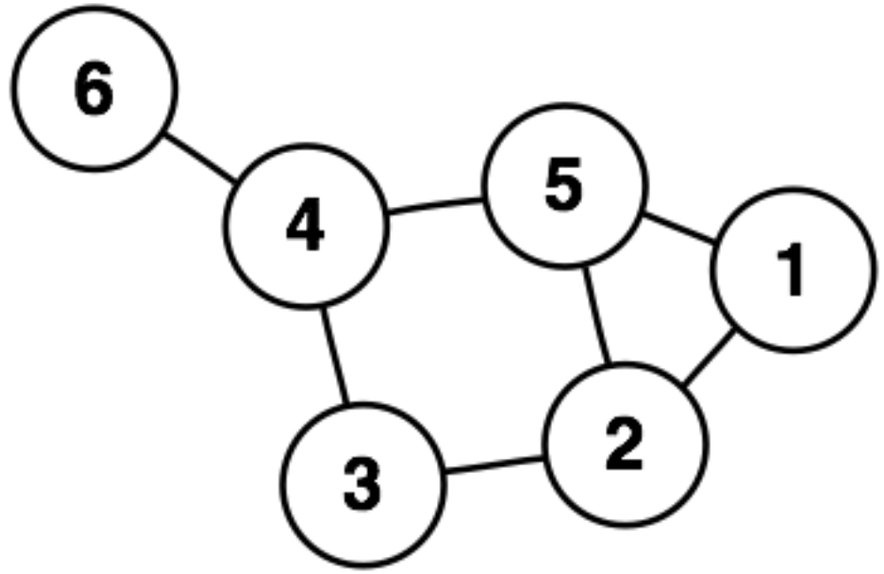


GRAPH



GRAPH

- A graph G = (V,E)
  - V = set of vertices
  - E = set of edges



-

SEARCHING ALGORITHM

INFORMED SEARCH VS UNINFORMED SEARCH

Uninformed Search

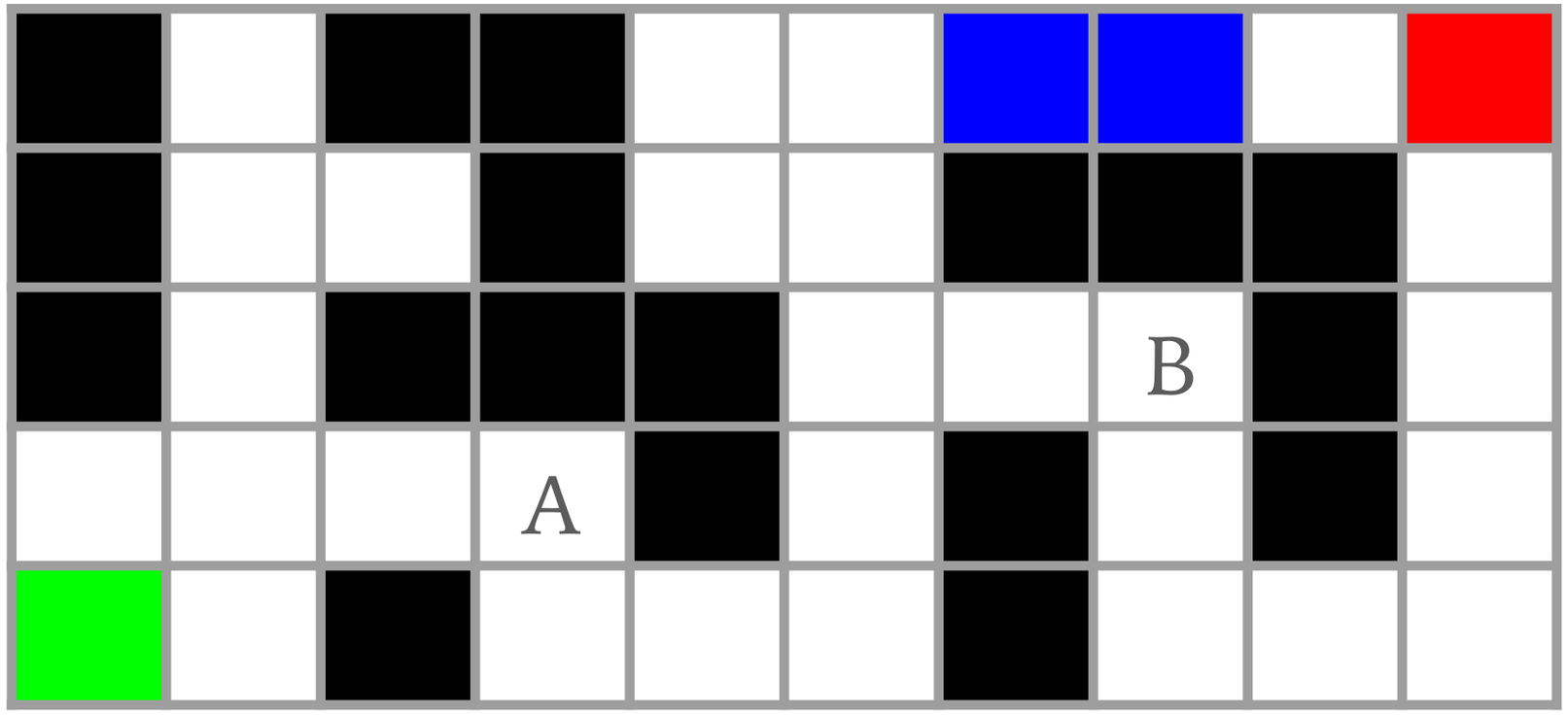
- Key Idea:
  - The algorithm doesn't know if state s is close to the goal state or not.

Informed Search

- Key Idea:
  - The algorithm know if state s is better than other state in order to reach the goal.

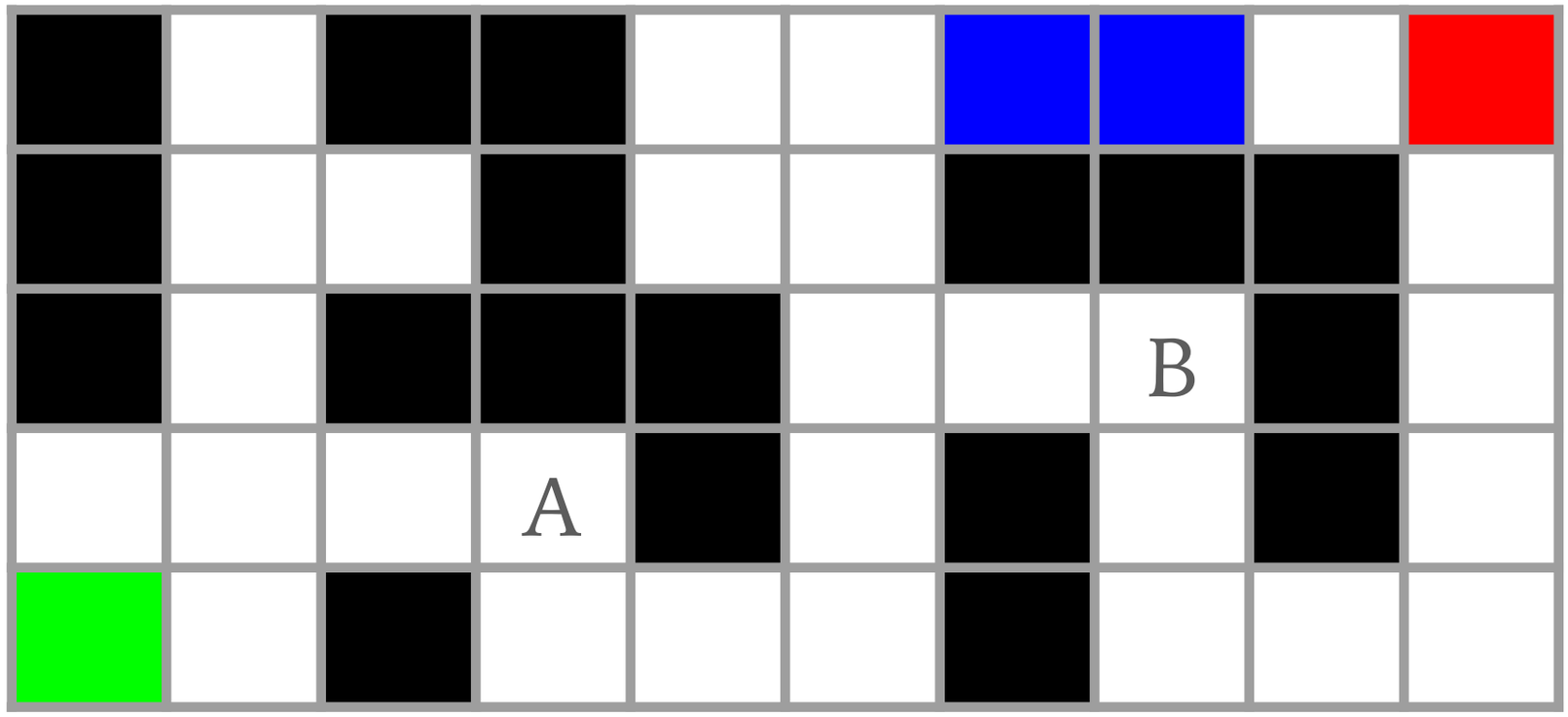
UNINFORMED SEARCH

- A robot at both states A and B can perform 2 actions: West, and South.
- It doesn't know if B is closer to the goal state or not.



INFORMED SEARCH

- The robot knows that state B is closer to the goal state than state A.
- It can take this information in to efficiently solve the problem.



UNINFORMED SEARCH

UNINFORMED SEARCH

- Key Idea:
  - The algorithm doesn't know if state s is close to the goal state or not.

- Definition
  - The strategies have no additional information about states beyond that provided in the problem definition.

BREADTH-FIRST SEARCH

Uninformed Search

BREADTH-FIRST SEARCH

Key Idea

- Expand first, then search, then expand, …

Definition

- 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.

BFS EXAMPLE

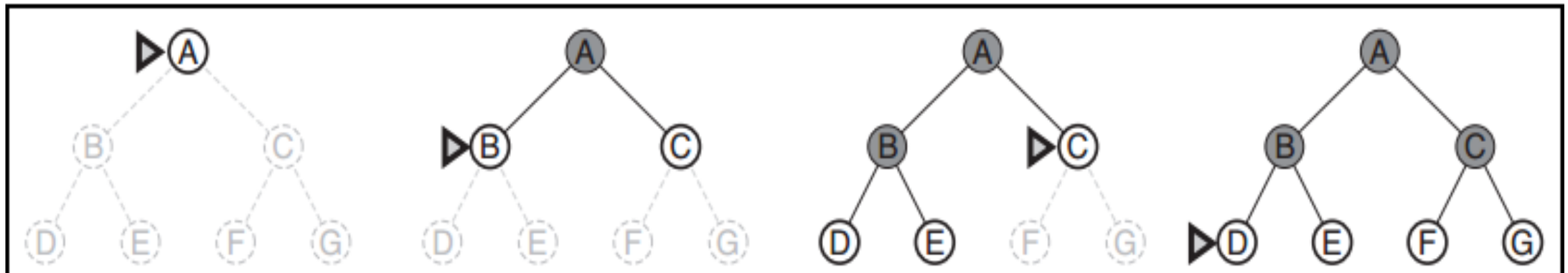


Figure 3.12 FILES: figures/bfs-progress.eps (Tue Nov 3 16:22:26 2009). Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

BFS: THE ALGORITHM

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

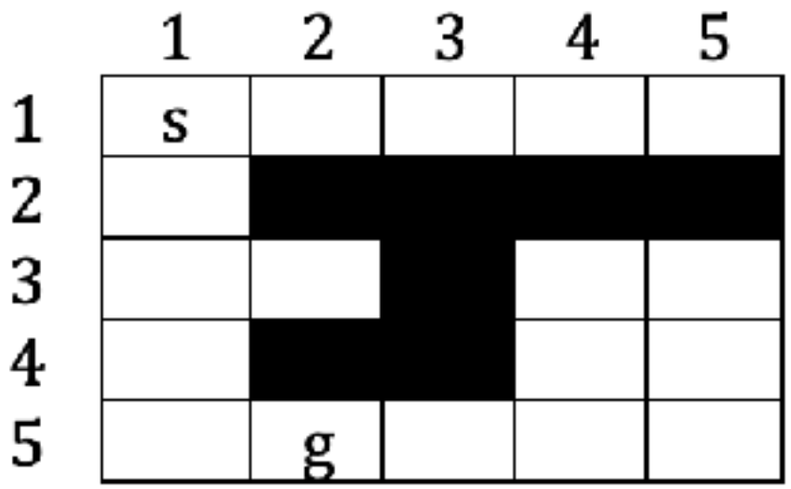
PROBLEMS WITH BREADTH-FIRST SEARCH

Given branching factor = 10, processing power = 1M nodes/sec, 1000 bytes/node:

Depth	Nodes	Time	Memory
2	110	.11 ms	107 kB
4	11110	11 ms	10.6 MB
10	10^10	3 hours	10 TB
16	10^16	350 years	10 EB

Required time and space increases exponentially.

BFS QUIZ



Starting state = (1,1)
Goal state = (2,5)

a. จงวาด tree ที่เกิดจากการทำ path planning โดยใช้ Breadth-first Search ให้ใส่ตัวเลขที่แต่ละกิ่ง (branch) เพื่อแสดงลำดับการแตกกิ่ง

b. Path ของการทำ path planning จาก Breath-first Search คืออะไร

UNIFORM-COST SEARCH

Uninformed Search

UNIFORM-COST SEARCH

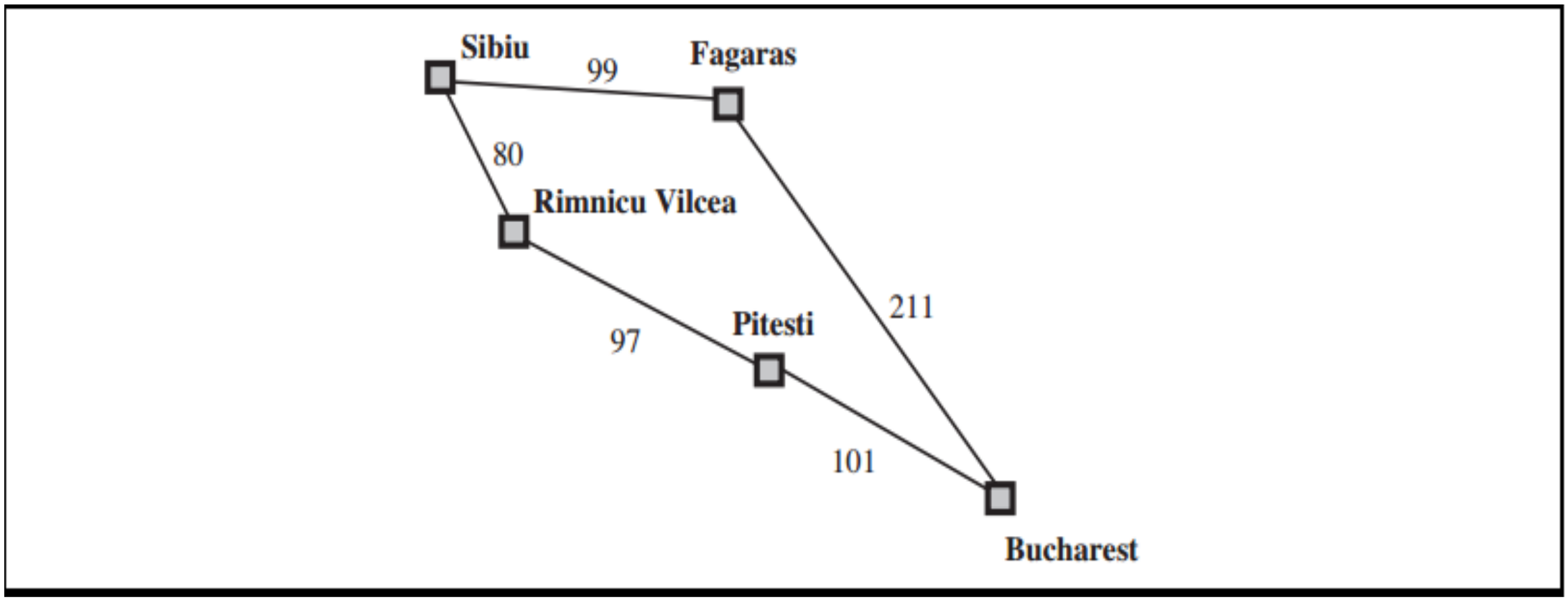
Key Idea:

- Don’t care about numbers of paths. Care about total cost.

Definition:

- Uniform-cost search expands the node n with the lowest path cost g(n) first.

UCS EXAMPLE

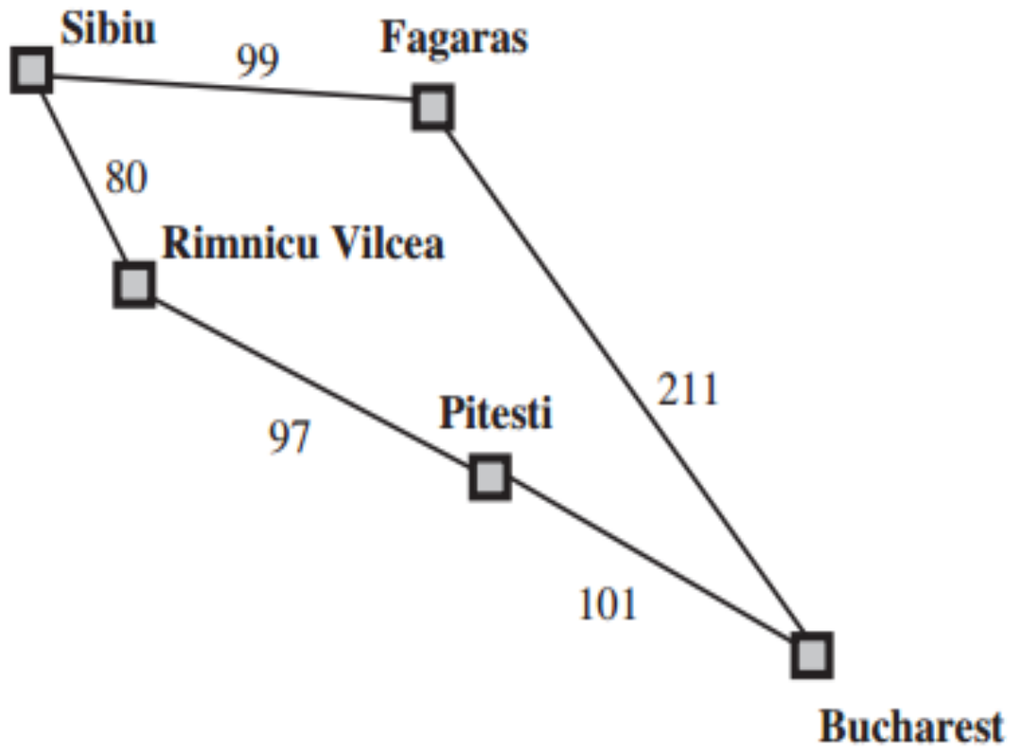

Sibiu
99
Fagaras
80
Rimnicu Vilcea
211
97
Pitesti
101
Bucharest

UCS EXAMPLE

Initial State: Sibiu

Goal State: Bucharest

Start at Sibui; cost = 0

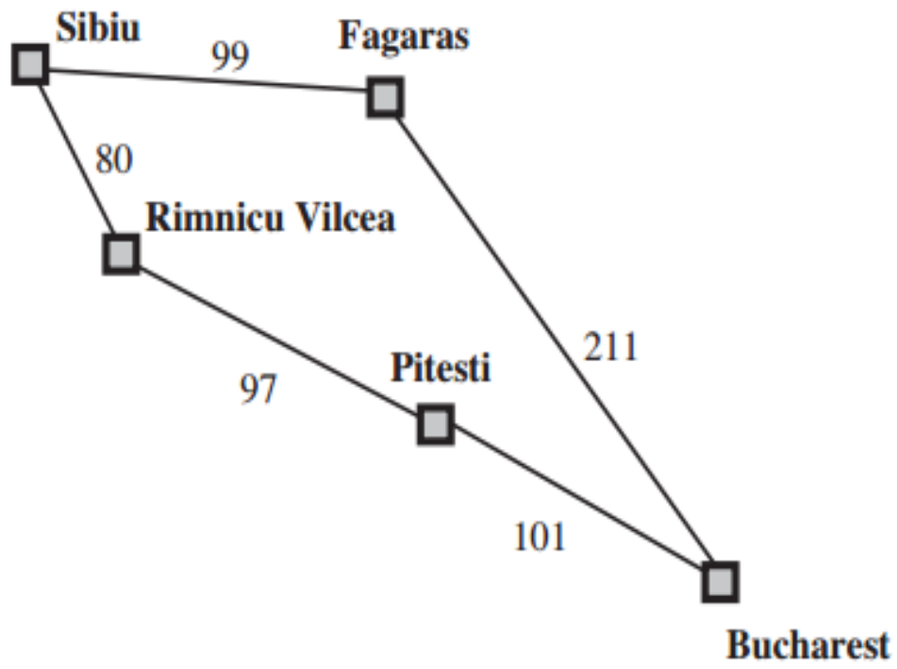


UCS EXAMPLE

First, expand root node.

Sibui =>Fagaras cost = 99

=>Rimnicu Vilcea cost = 80

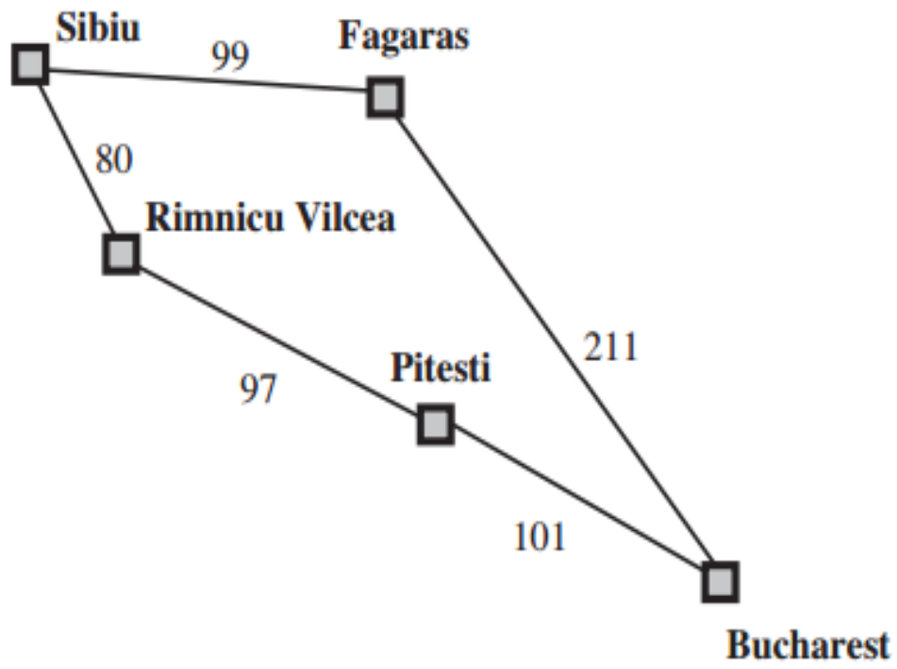


UCS EXAMPLE

Next, expand the lower cost path.

Sibui =>Fagara cost = 99

=>Rimnicu Vilcea=>Pitesti cost = 80 + 97 = 177

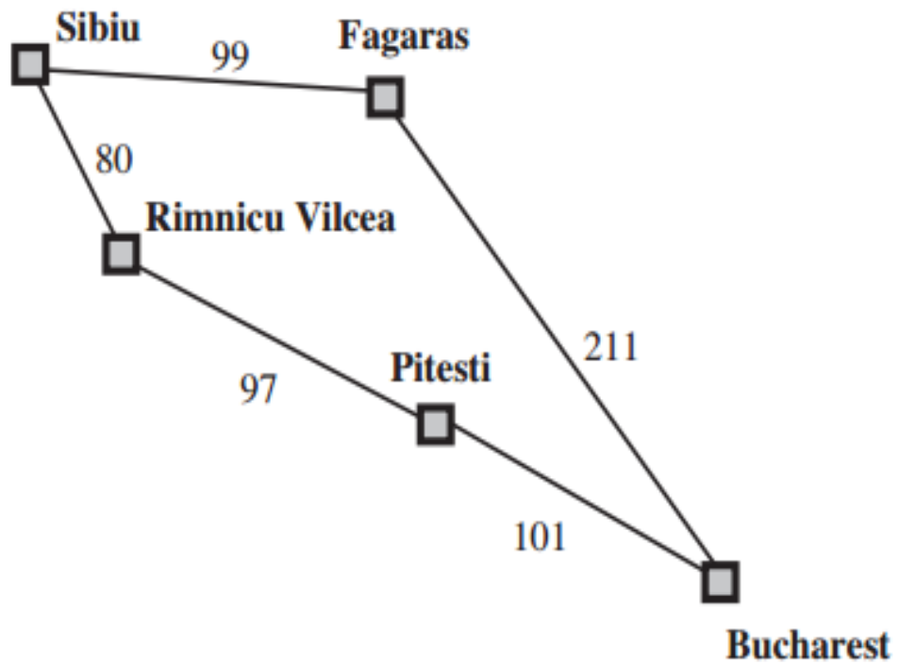


UCS EXAMPLE

Keep expanding the lower cost path.

Sibui =>Fagaras=>Bucharest cost = 99 + 211 = 310

=>Rimnicu Vilcea=>Pitesti cost = 177



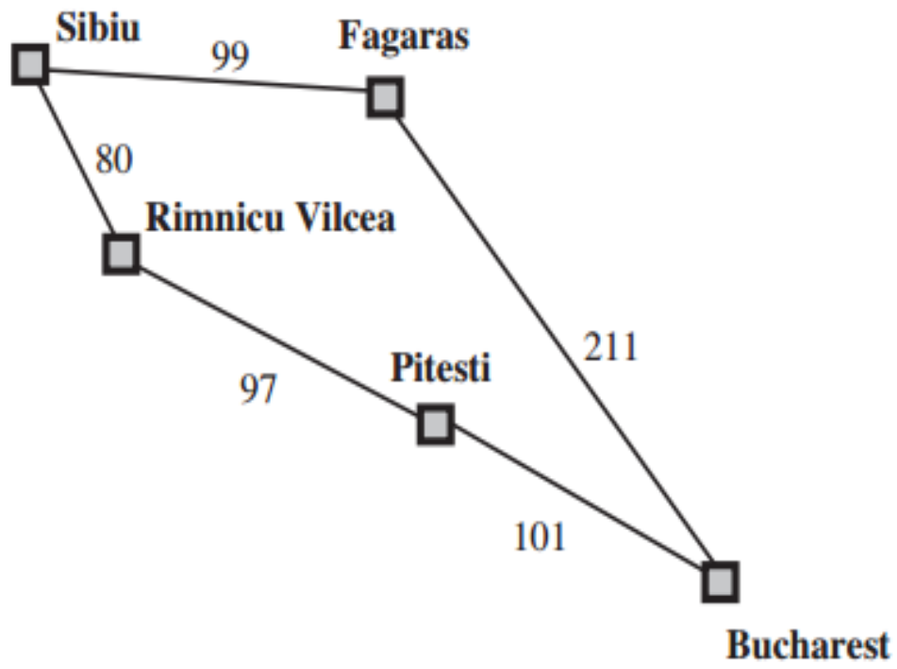
UCS EXAMPLE

Keep expanding the lower cost path.

Sibui =>Fagaras=>Bucharest cost = 310

=>Rimnicu Vilcea=>Pitesti=>Bucharest

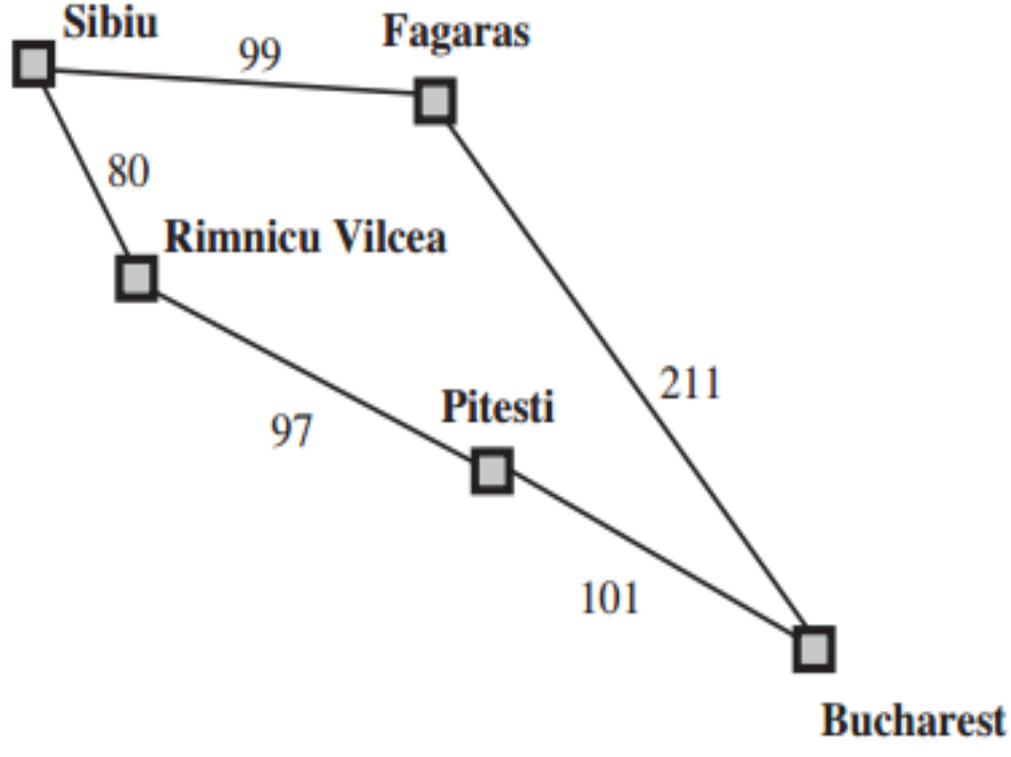
cost = 177 + 101= 278



UCS EXAMPLE

Goal State is reached. Answer is the lowest cost path:

Sibui => Rimnicu Vilcea => Pitesti => Bucharest
cost = 278



UCS: THE ALGORITHM

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

DEPTH-FIRST SEARCH

Uninformed Search

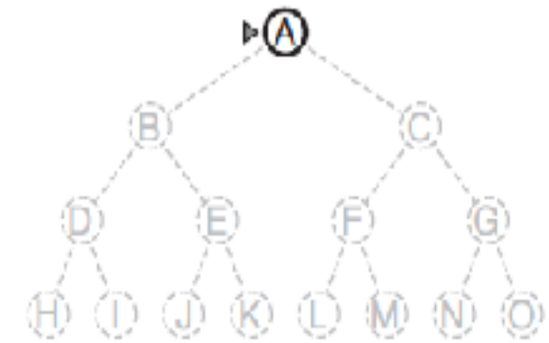
DEPTH-FIRST SEARCH

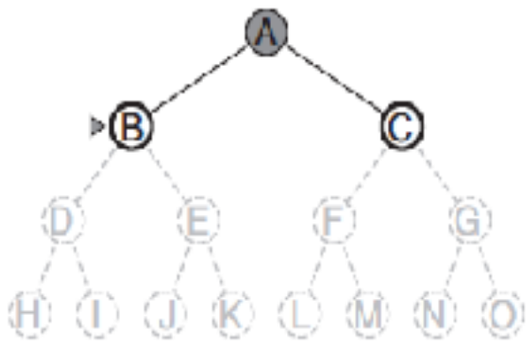
Key Idea:

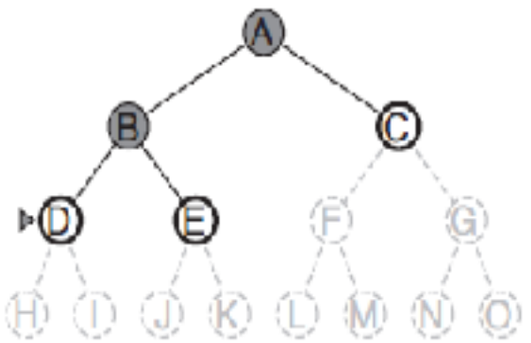
- Expand the lowest node first..

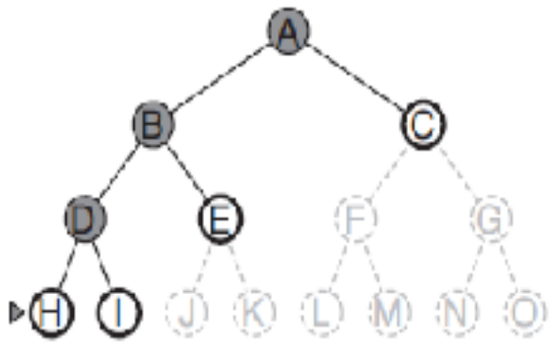
Definition:

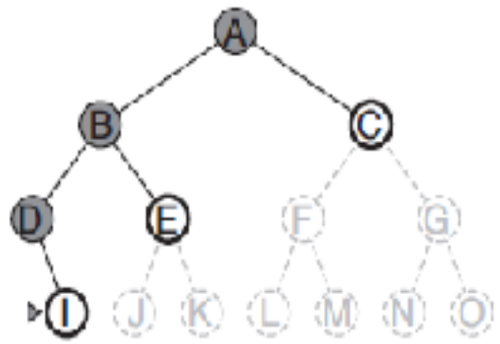
- Depth-first Search expands the deepest node in the current frontier of the search tree.

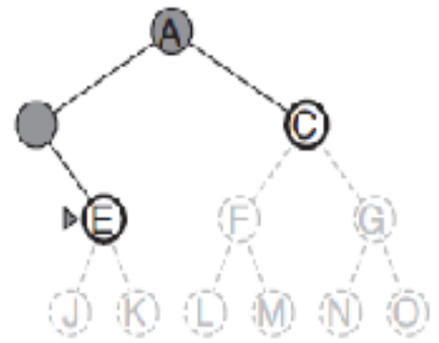

A
B C
D E F G
H I J K L M N O

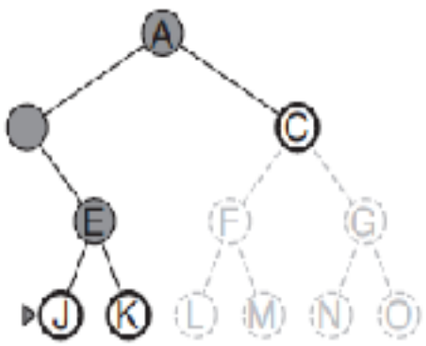

A
B C
D E F G
H I J K L M N O

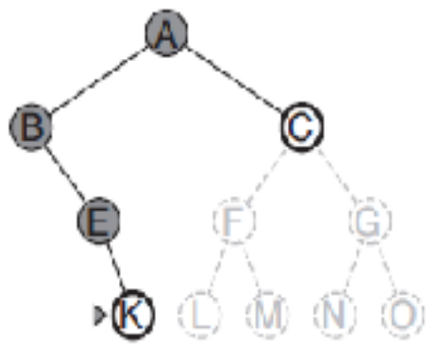

A
B C
D E F G
H I J K L M N O

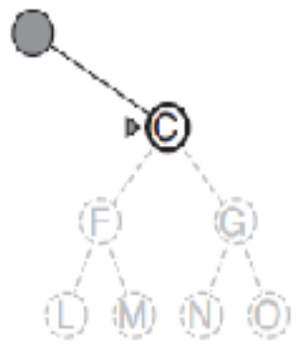

A
B C
D E F G
H I J K L M N O

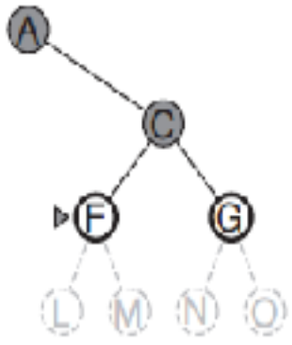

A
B C
D E F G
I J K L M N O

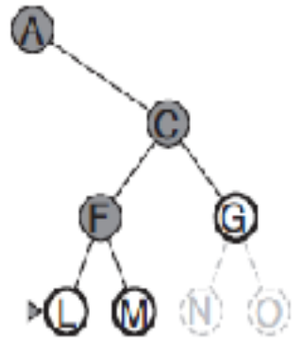

A
C
E F G
J K L M N O

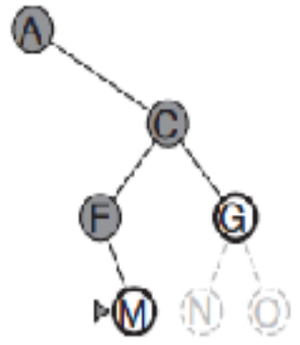

A
C
E F G
J K L M N O


A
B C
E F G
K L M N O


C
F G
L M N O


A
C
F G
L M N O


A
C
F G
L M N O

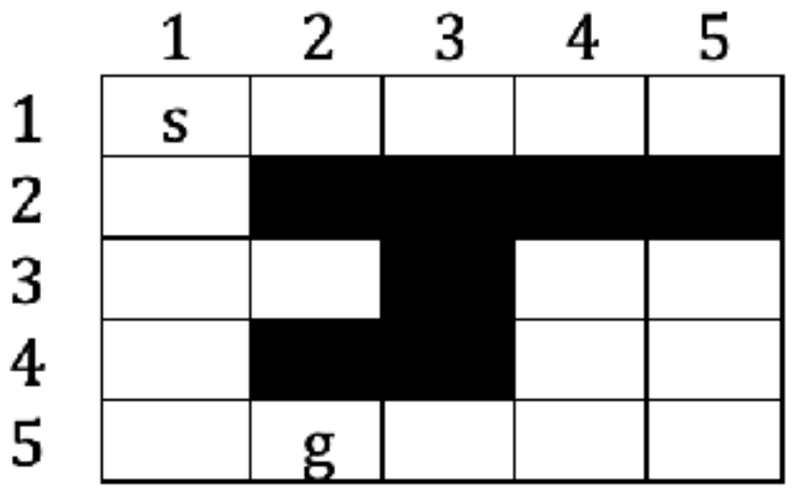

A
C
F G
M N O

PROBLEM WITH DEPTH-FIRST SEARCH

Searching in infinite spaces

- What if a child node of A is B and C, and a child node of B is A and D.
- We will have a tree that looks like:

DFS QUIZ



Starting state = (1,1)
Goal state = (2,5)

c. กำหนดให้ลำดับการแตก neighbor เป็นดังนี้: east, south, west, north, จงวาด tree ที่เกิดจากการทำ path planning โดยใช้ Depth-first Search ให้ใส่ตัวเลขที่แต่ละกิ่ง(branch) เพื่อแสดงลำดับการแตกกิ่ง

d. Path ของการทำ path planning จาก Depth-first Search คืออะไร

DEPTH-LIMITED SEARCH

Uninformed Search

DEPTH-LIMITED SEARCH

Key Idea:

- Same as Depth-first search, with limit on the depth (maximum level to expand)

Definition:

- Depth-limited Search is supplied with a predetermined depth limit l. Nodes at depth l are treated as if they have no successors.

DLS: THE ALGORITHM

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

PROBLEMS WITH DLS

How do we pick the depth limit?

- We need to supply a depth limit l to the algorithm, so the algorithm terminates at some points.
- What if the goal state is at depth l + 1 ?

ITERATIVE DEEPENING DEPTH-FIRST SEARCH

Uninformed Search

ITERATIVE DEEPENING DEPTH-FIRST SEARCH

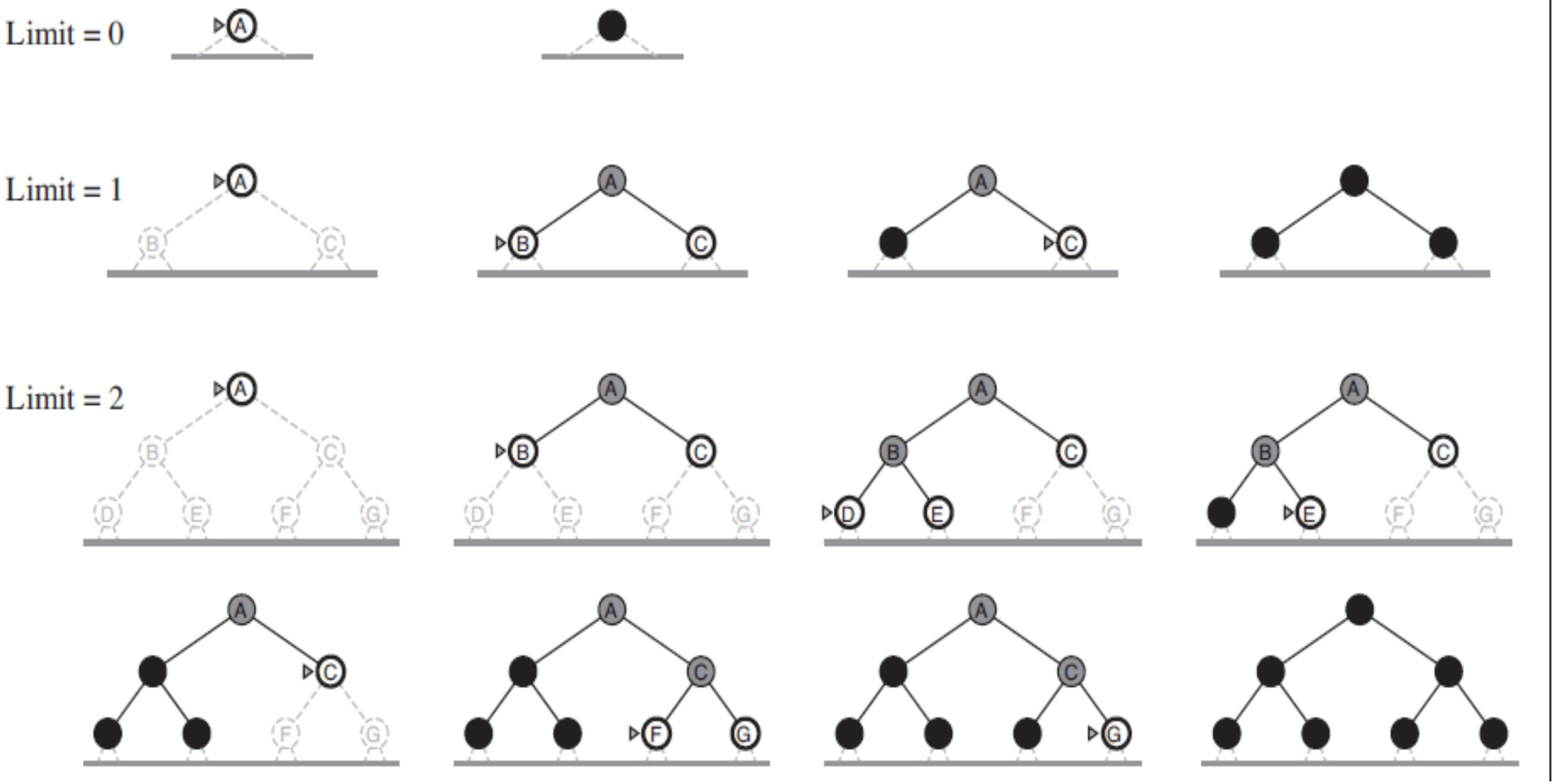
Key Idea:

- Iterating DFS with increasing depth limit.

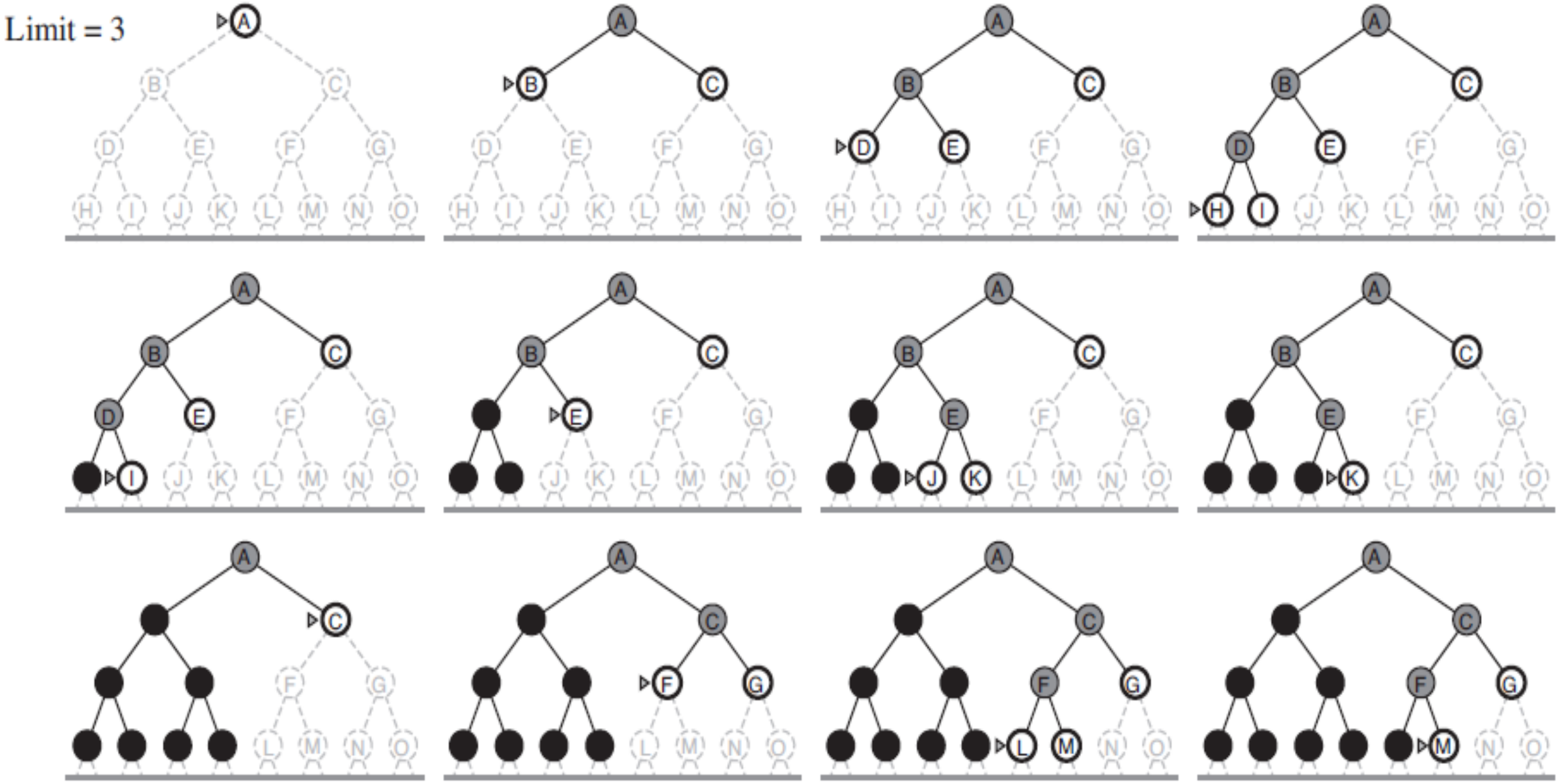
Definition:

- Iterative Deepening Depth-first Search gradually increases the limit - first 0, then 1, then 2, and so on - until the goal is found.

IDDFS: EXAMPLE



IDDFS: EXAMPLE



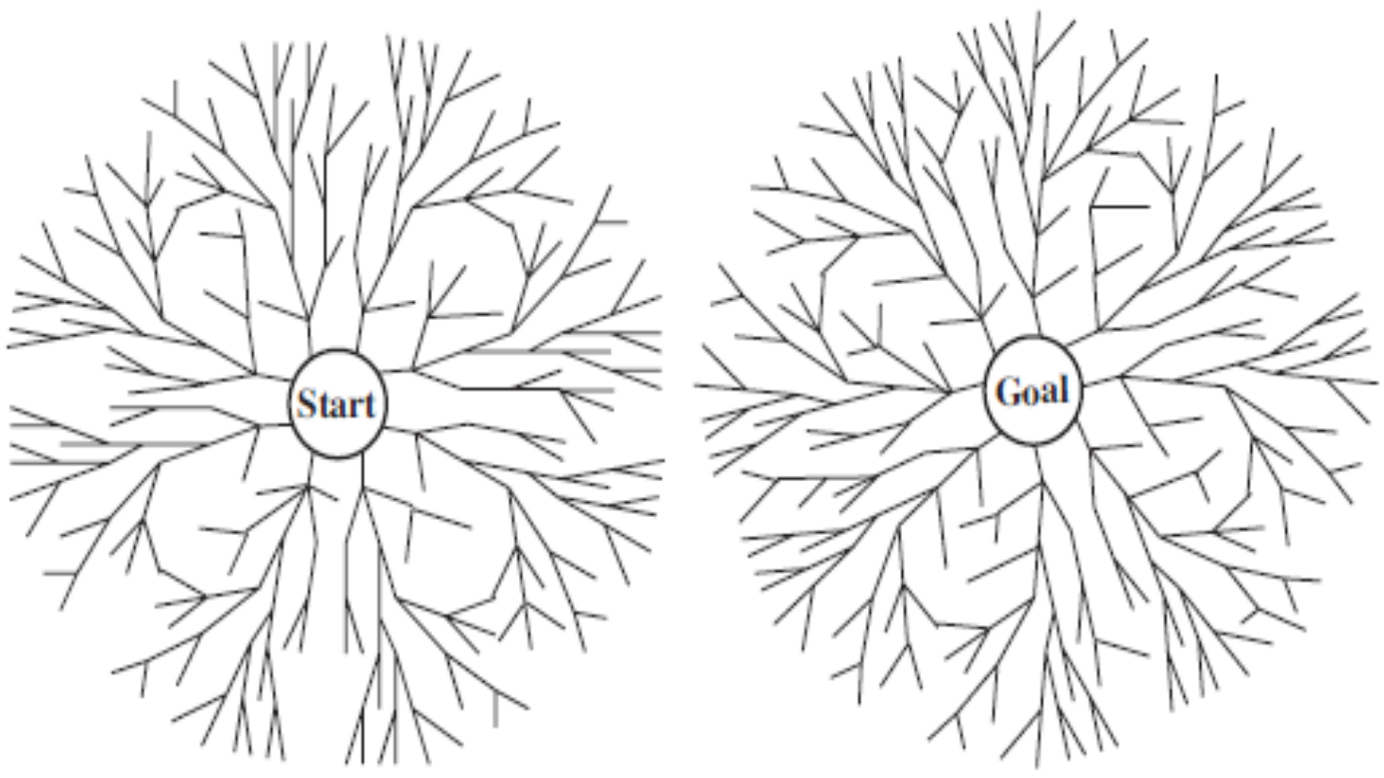
IDDFS: THE ALGORITHM

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* $\neq$ cutoff **then return** *result*

OTHER ALGORITHM

Bidirectional Search

- Two simultaneous searches: one from start state, one from goal state
- Check if expanded nodes are overlapped



LAB TIME

- Problem Set 1 will consist of 3 Problem Solving minilab.
- Go to github.com/BawornsakS/FRA311
- Access Problem Solving Lab 1 and follow the instruction.
- You can write your answer and submit it via Google Classroom.

PATH PLANNING & OPTIMIZATION PROBLEM

OPTIMIZATION

- According to Google Translate, optimization is “the action of making the best or most effective use of a situation or resource.”

- Basically, finding the best moves to perform, so it can lead to the goal efficiently.

ADVANCED DATA STRUCTURE

PRIORITY QUEUE

- A queue (FIFO) with a priority.
  - We can construct it in a form of (element, priority).

- Given a queue:
  - (A,1), (B,4), (Z,7), (B, 10)

- If we want to add another node (C, 5)
  - Normal queue will output:
  - Priority queue will output:

PRIORITY QUEUE

- A queue (FIFO) with a priority.
  - We can construct it in a form of (element, priority).

- Given a queue:
  - (A,1), (B,4), (Z,7), (B, 10)

- If we want to add another node (C, 5)
  - Normal queue will output: (A,1), (B,4), (Z,7), (B, 10), (C, 5)
  - Priority queue will output:

PRIORITY QUEUE

- A queue (FIFO) with a priority.
  - We can construct it in a form of (element, priority).

- Given a queue:
  - (A,1), (B,4), (Z,7), (B, 10)

- If we want to add another node (C, 5)
  - Normal queue will output: (A,1), (B,4), (Z,7), (B, 10), (C, 5)
  - Priority queue will output: (A,1), (B,4), (C, 5), (Z,7), (B, 10)

HEURISTIC SEARCH

Informed Search

INFORMED SEARCH VS UNINFORMED SEARCH

Uninformed Search

- Key Idea:

  The algorithm doesn't know if state s is close to the goal state or not.

Informed Search (Heuristic Search)

- Key Idea:

  The algorithm know if state s is better than other state in order to reach the goal

HEURISTIC SEARCH

Key Idea:

The algorithm know if state s is better than other state in order to reach the goal

Definition:

Informed Search (Heuristic Search) uses problem-specific knowledge beyond the definition of the problem itself to determine the best path to the goal

BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- Let's define what the problem specific knowledge are.

- In problem formation from the last lecture, what do we know?

BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- Let's define what the problem specific knowledge are.

- In problem formation from the last lecture, what do we know?
  - Initial state
  - Goal state
  - All states
  - Actions
  - Result(s,a)
  - Cost(s,a)

BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- In problem formation from the last lecture, what do we know?
  - Initial state, goal state, all states, actions, result(s,a), cost(s,a)

- From these information, can we derive some additional information from them?

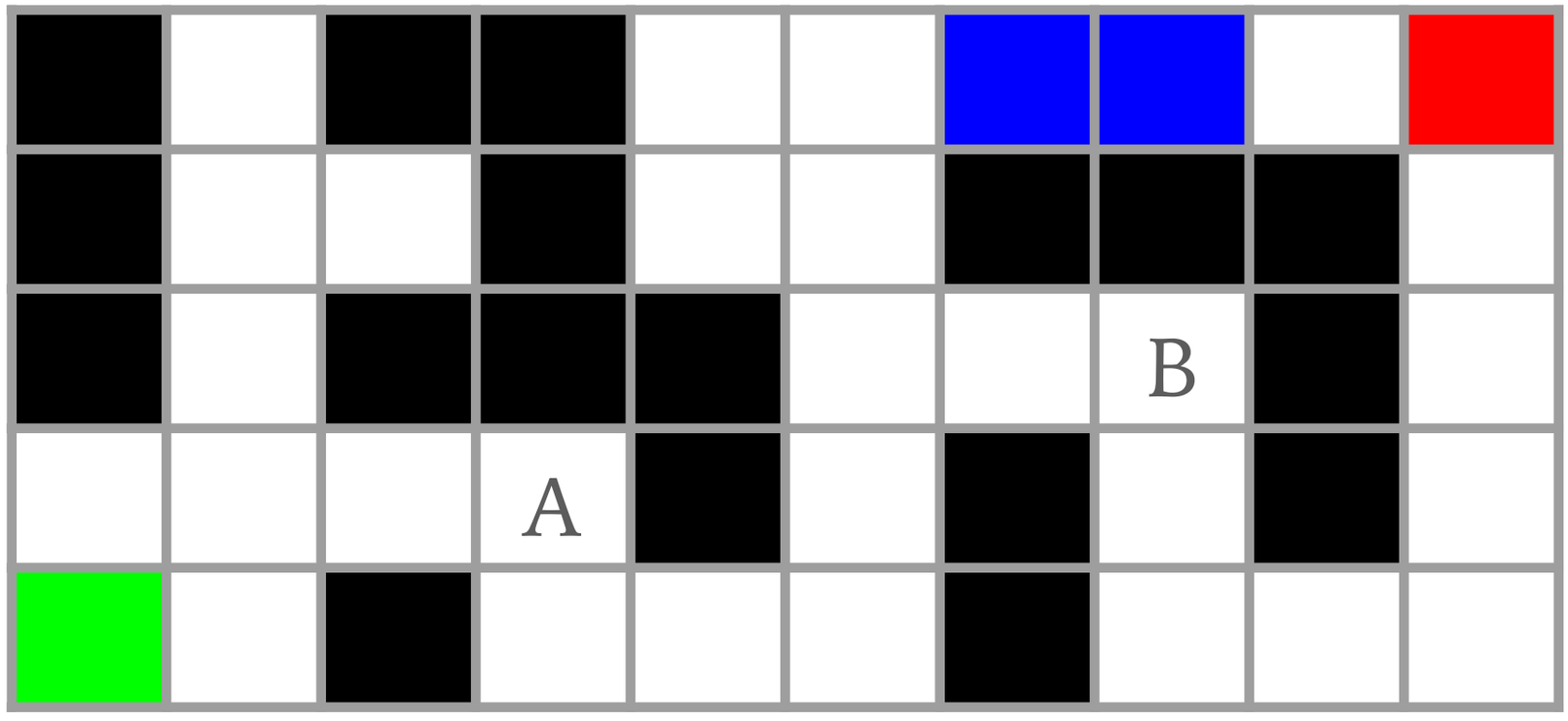
BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- In problem formation from the last lecture, what do we know?
  - Initial state, goal state, all states, actions, result(s,a), cost(s,a)

- From these information, can we derive some additional information from them?
- What if the world is a grid, and we want to find how far we are from the goal state?

BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- If we want to know how far A and B are from the goal (red), how do we do it?



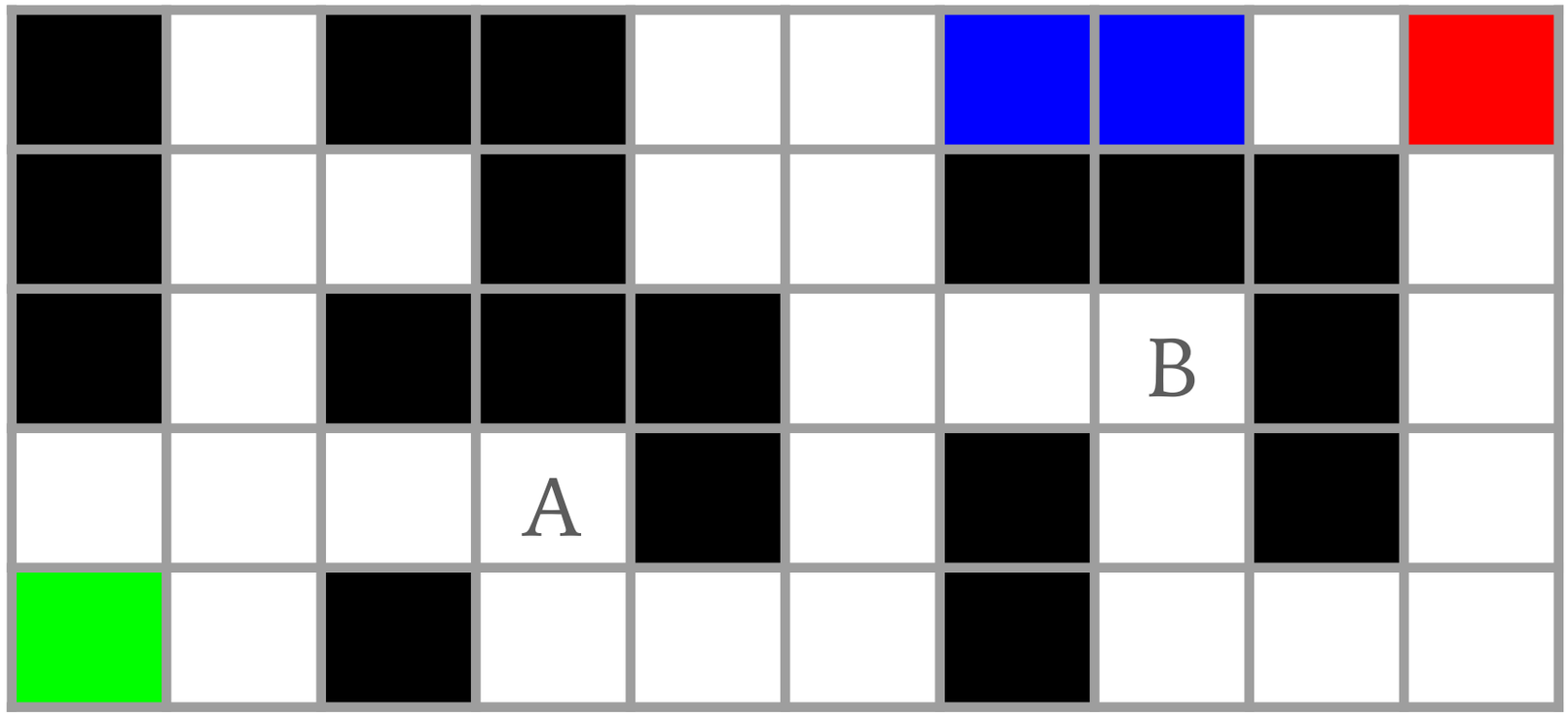
HEURISTIC FUNCTION

- For Informed Search, we use a Heuristic Function, h(n), to determine the node to expand..

- Definition:
  - Heuristic function estimates cost of the cheapest path from the state at node n to a goal state.

BEYOND PROBLEMS SPECIFIC KNOWLEDGE

- What are the heuristic functions you can use on this map?



DISTANCE FUNCTION

- Generally, one popular heuristic function is a distance function.

- A distance function is a function to calculate a distance between two points in n dimensional space.

- Two most popular distance functions are
  - Manhattan distance
  - Euclidean distance

MANHATTAN DISTANCE

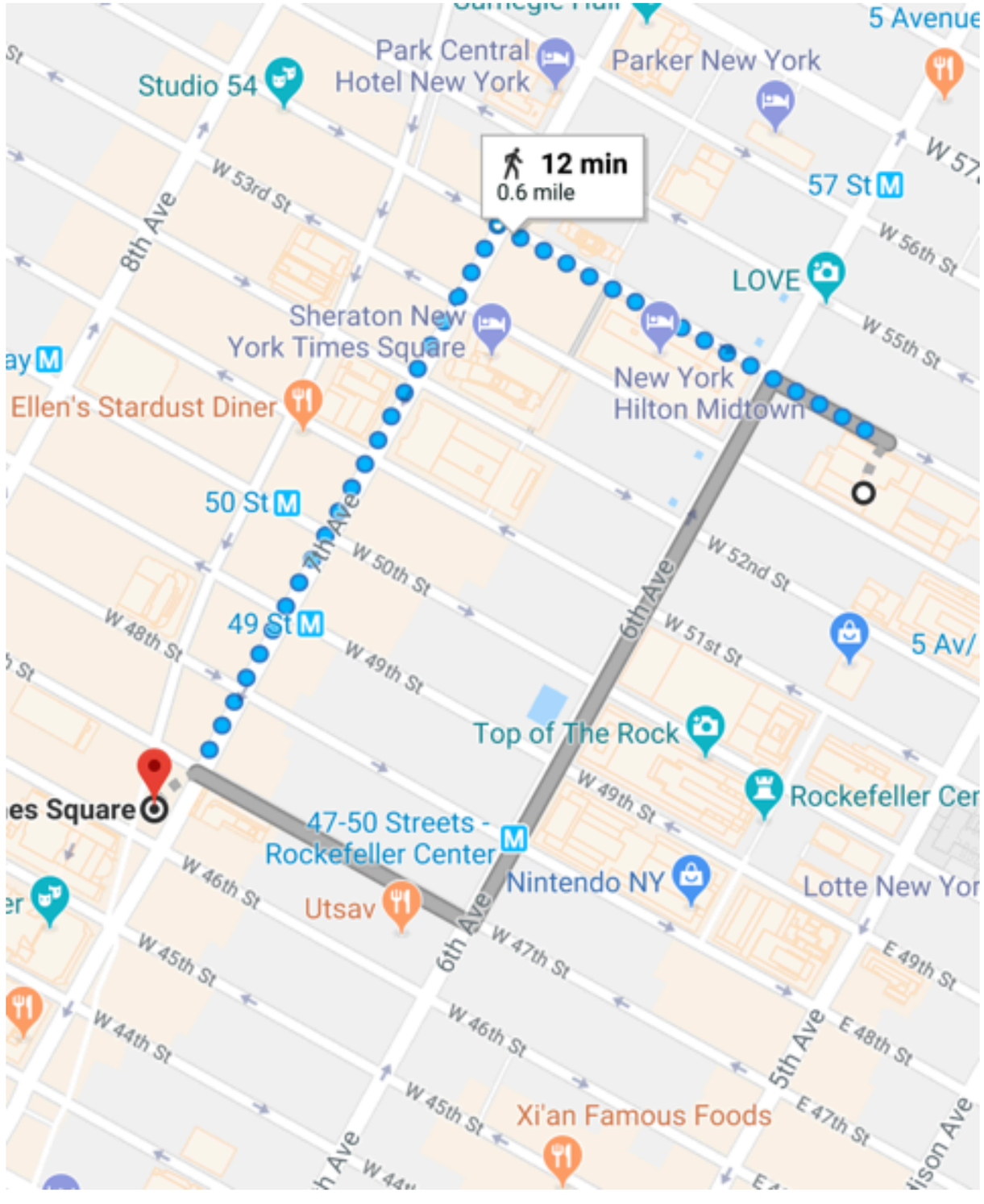
- The name is from a part of New York City: Manhattan.

MANHATTAN DISTANCE

- The name is from a part of New York City: Manhattan.

MANHATTAN DISTANCE

- If we want to walk from one place to another, we need to walk along the street.
  - And people count it as a "block".
- Like, go north 5 for blocks, then east 3 for blocks.



MANHATTAN DISTANCE

- In 2-d:

$$d = \Delta x_1 + \Delta x_2$$

- In n-d:

$$d = \Delta x_1 + \Delta x_2 + ... + \Delta x_n$$

EUCLIDEAN DISTANCE

- A fancy name of what we call “straight line distance”

- Can you guess what a straight line distance is?

EUCLIDEAN DISTANCE

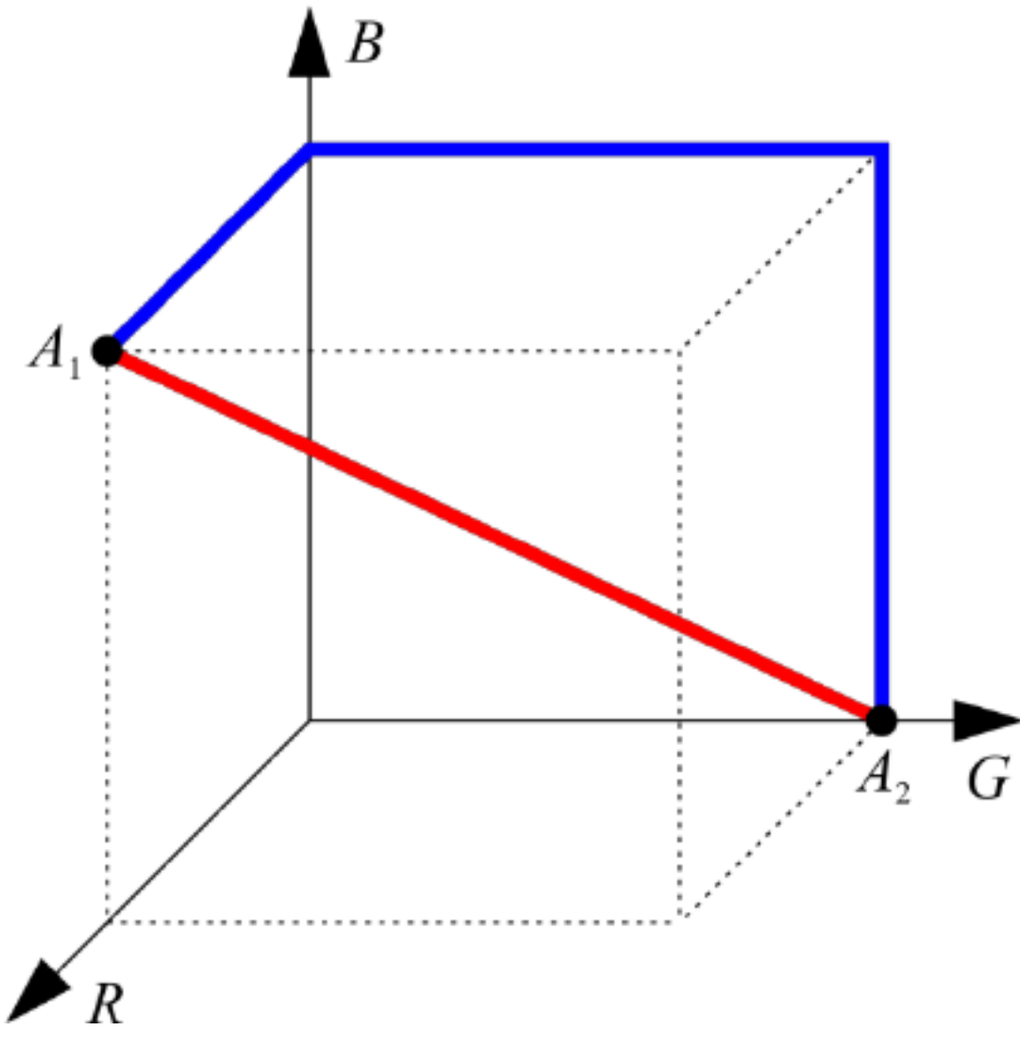
- In 2-d:

$$d = \sqrt{\Delta x_1^2 + \Delta x_2^2}$$

- In n-d:

$$d = \sqrt{\Delta x_1^2 + \Delta x_2^2 + ... + \Delta x_n^2}$$

MANHATTAN VS EUCLIDEAN



Retrieved from : https://www.researchgate.net/profile/Katrina_Bolochko/publication/273487115/figure/fig3/AS:284032651808772@1444730069527/Fig-3-Manhattan-and-Euclidean-distances-between-two-points.png

GREEDY BEST-FIRST SEARCH

Informed Search

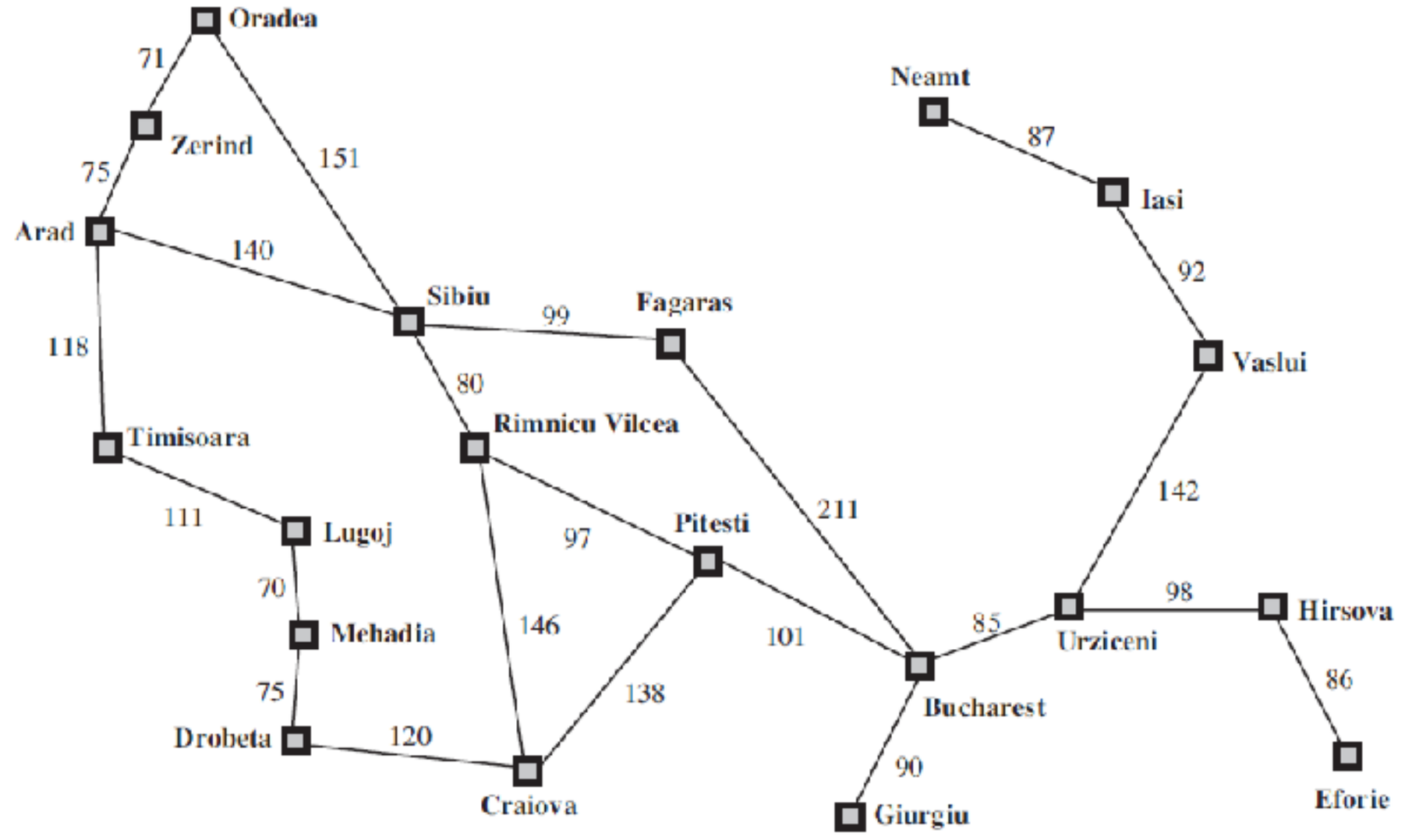
GREEDY BEST-FIRST SEARCH

Key idea:

- Expand the node that is the nearest to the goal first

Definition:

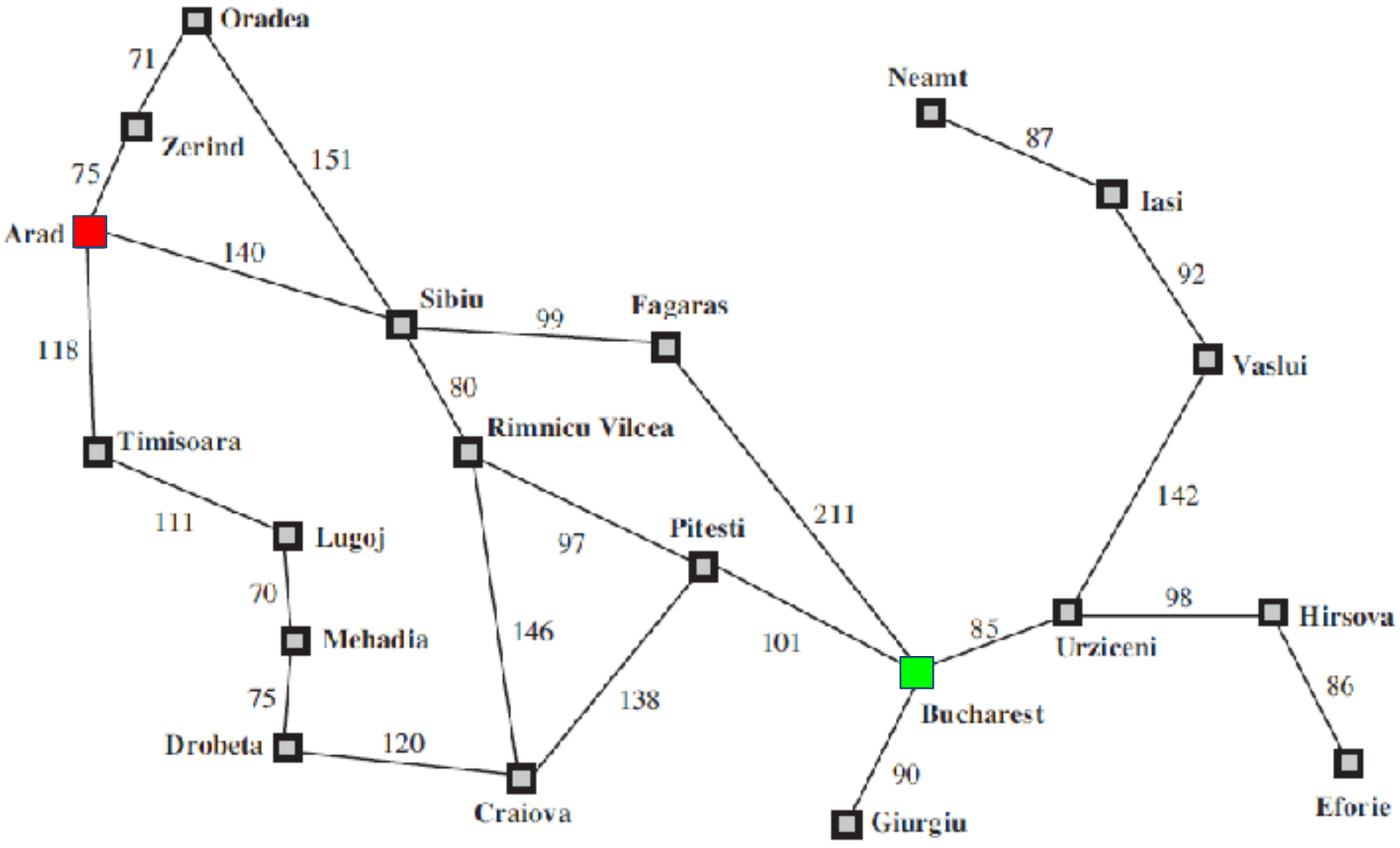
- Greedy Best-first Search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.



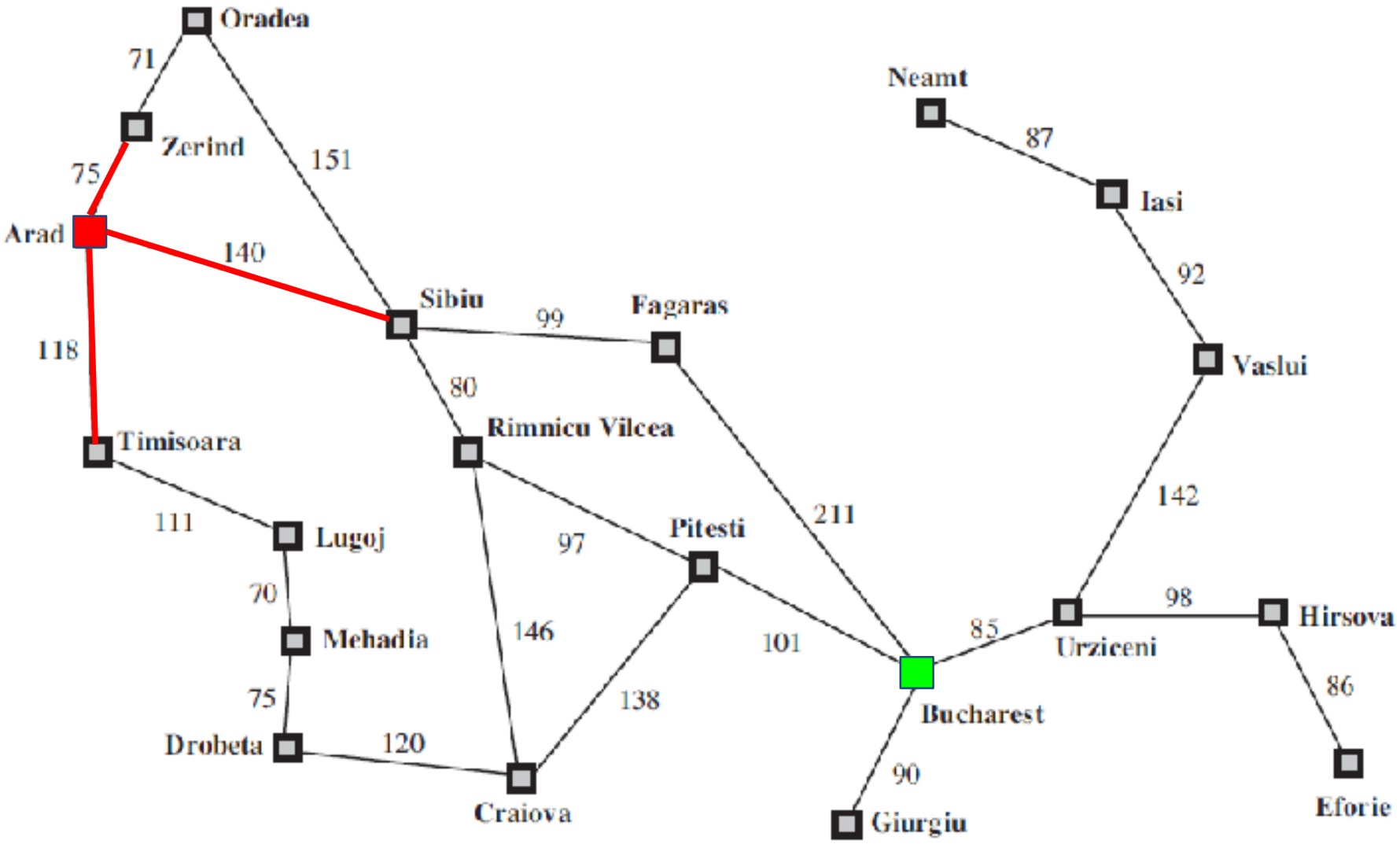
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

HEURISTIC FUNCTION

h(n) = straight line distance from node n to the goal state.

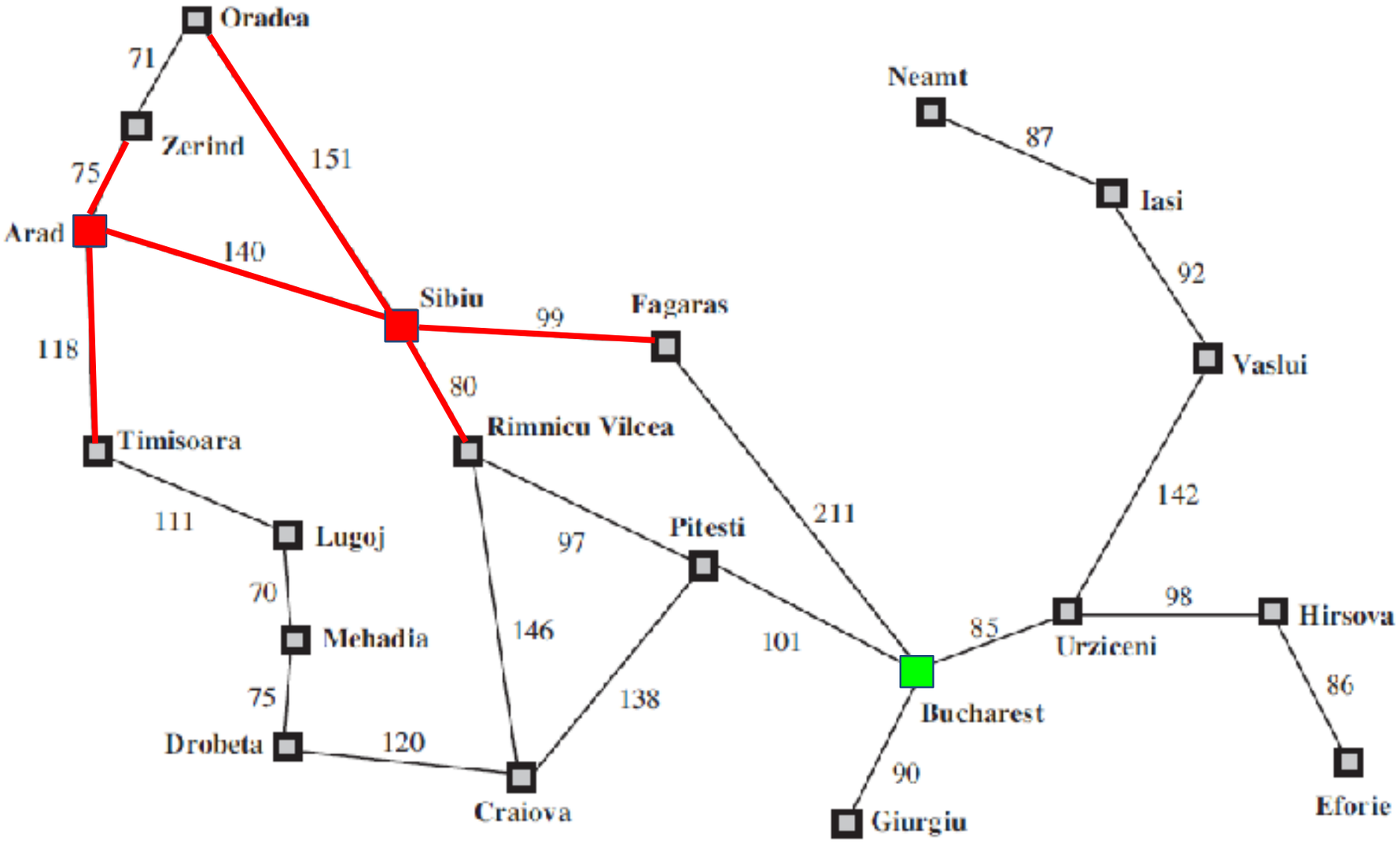


Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



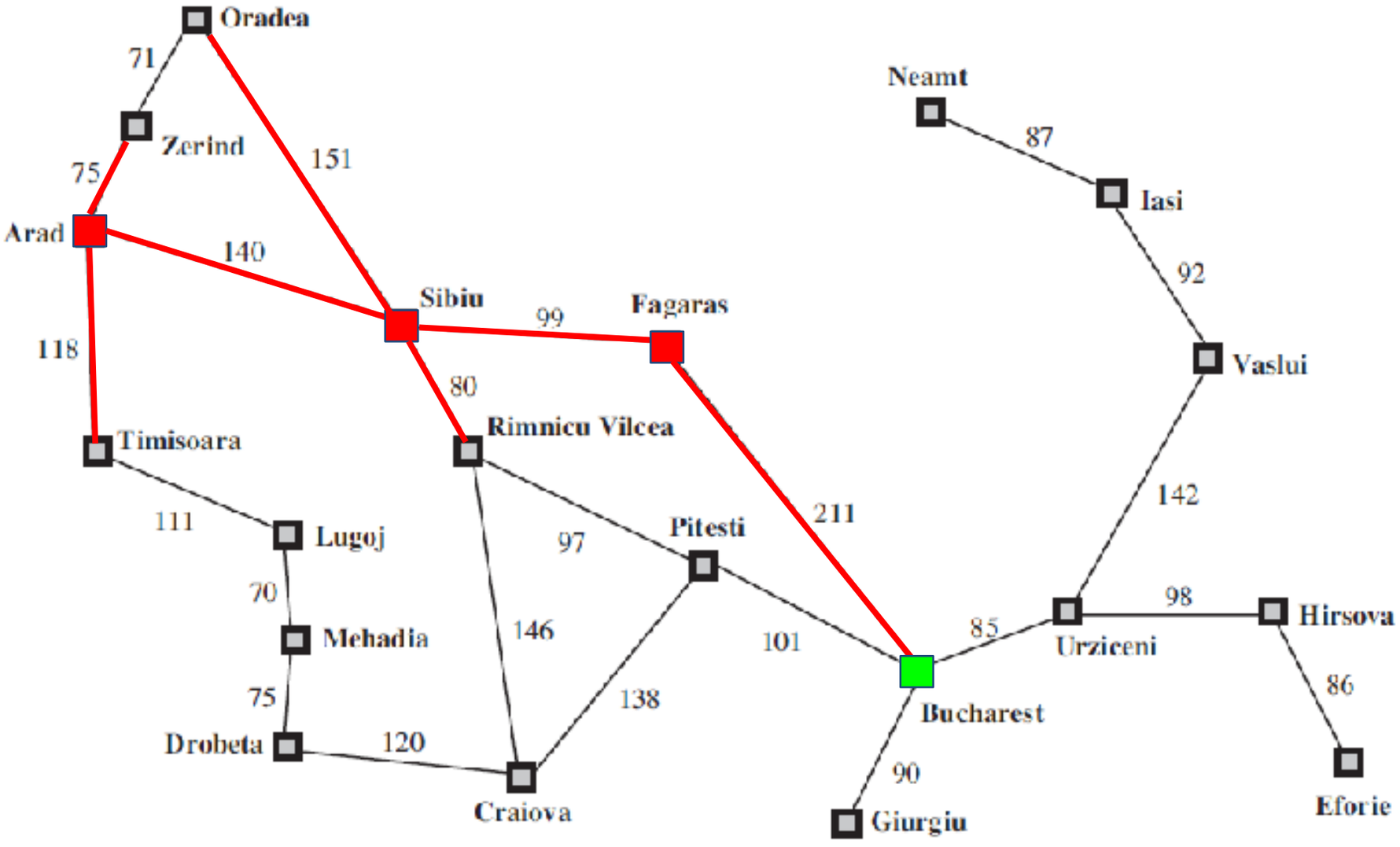
...

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



...

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



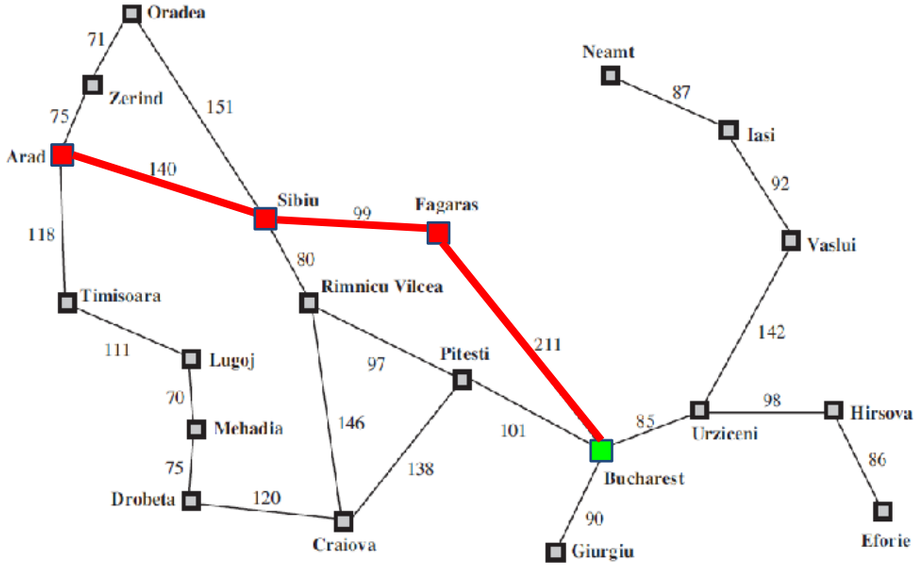
...

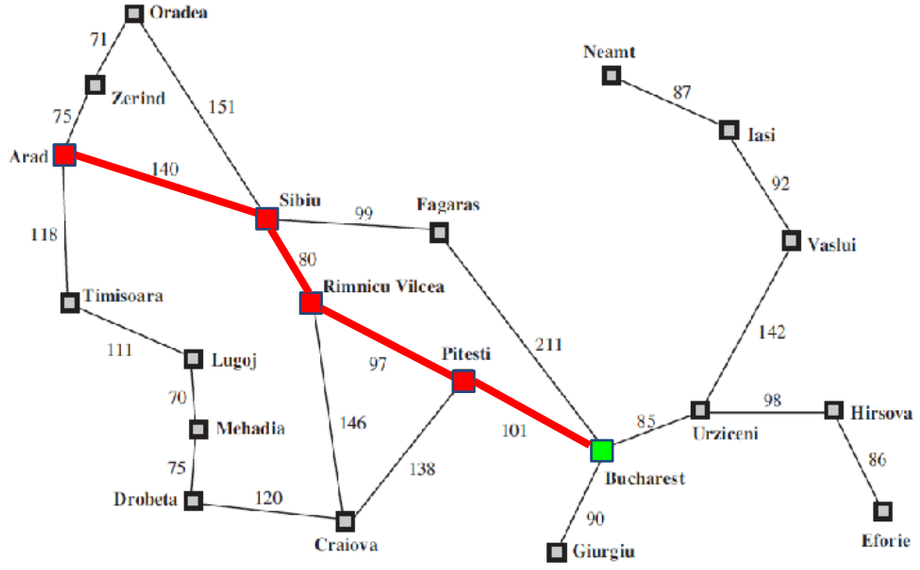
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

GREEDY BEST-FIRST SEARCH

It looks good, but it is not optimal.

- Path on the left is 32 kilometers longer.





A* SEARCH

Informed Search

A* SEARCH (A-STAR SEARCH)

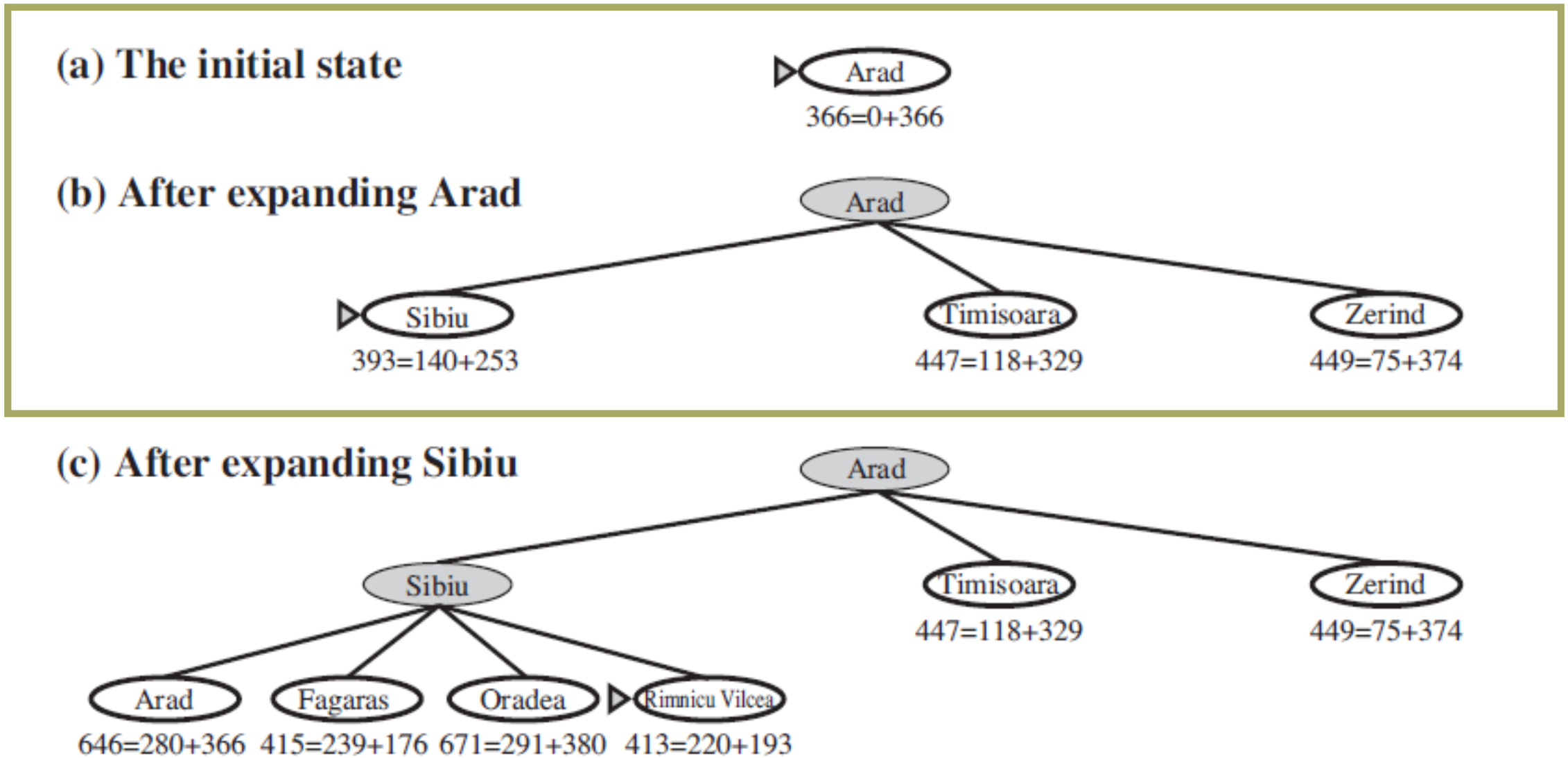
- Key Idea:
  - Expands the cheapest node
  - Cheapest node = minimum cost of (initial state to node n + node n to goal state)

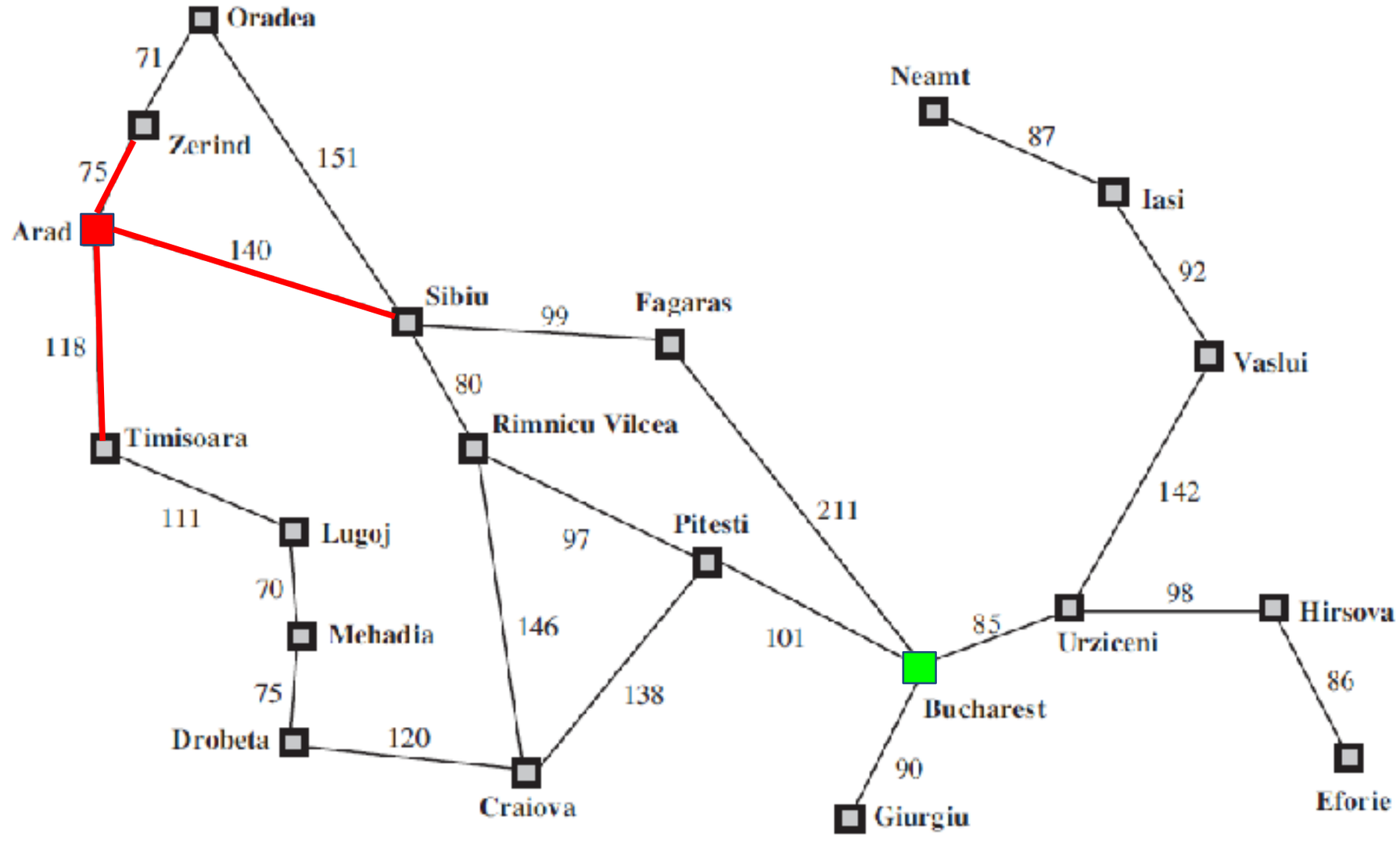
- Definition:
  - A* Search is a form of best-first search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n) the cost to get from the node to the goal:
  - $f(n) = g(n) + h(n)$

EVALUATION FUNCTION

- A* Search is a form of best-first search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n) the cost to get from the node to the goal:

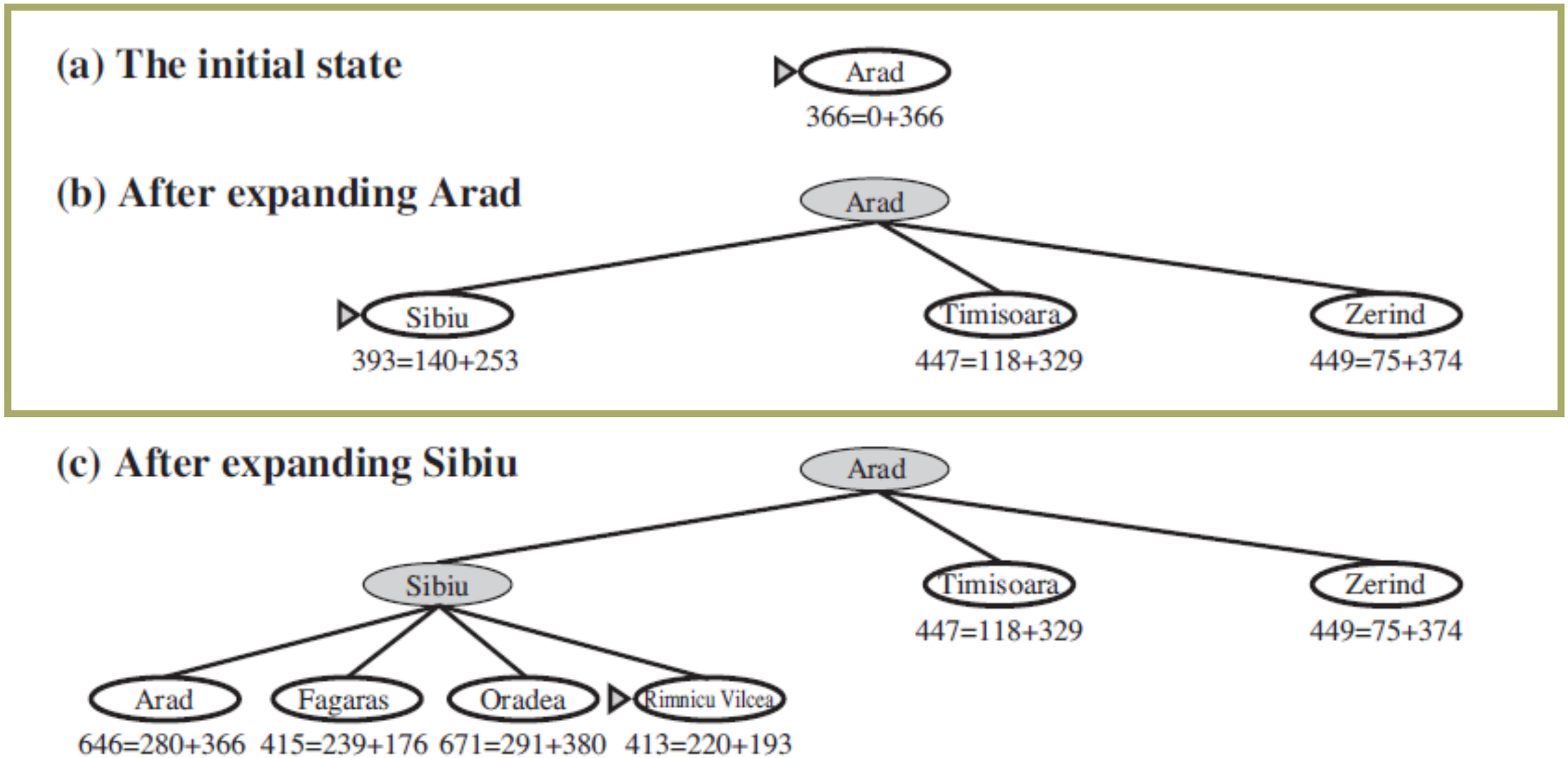
- $f(n) = g(n) + h(n)$
  - f(n) = Evaluation function
  - g(n) = Cost function from initial state to current state
  - h(n) = Heuristic function from current state to goal state

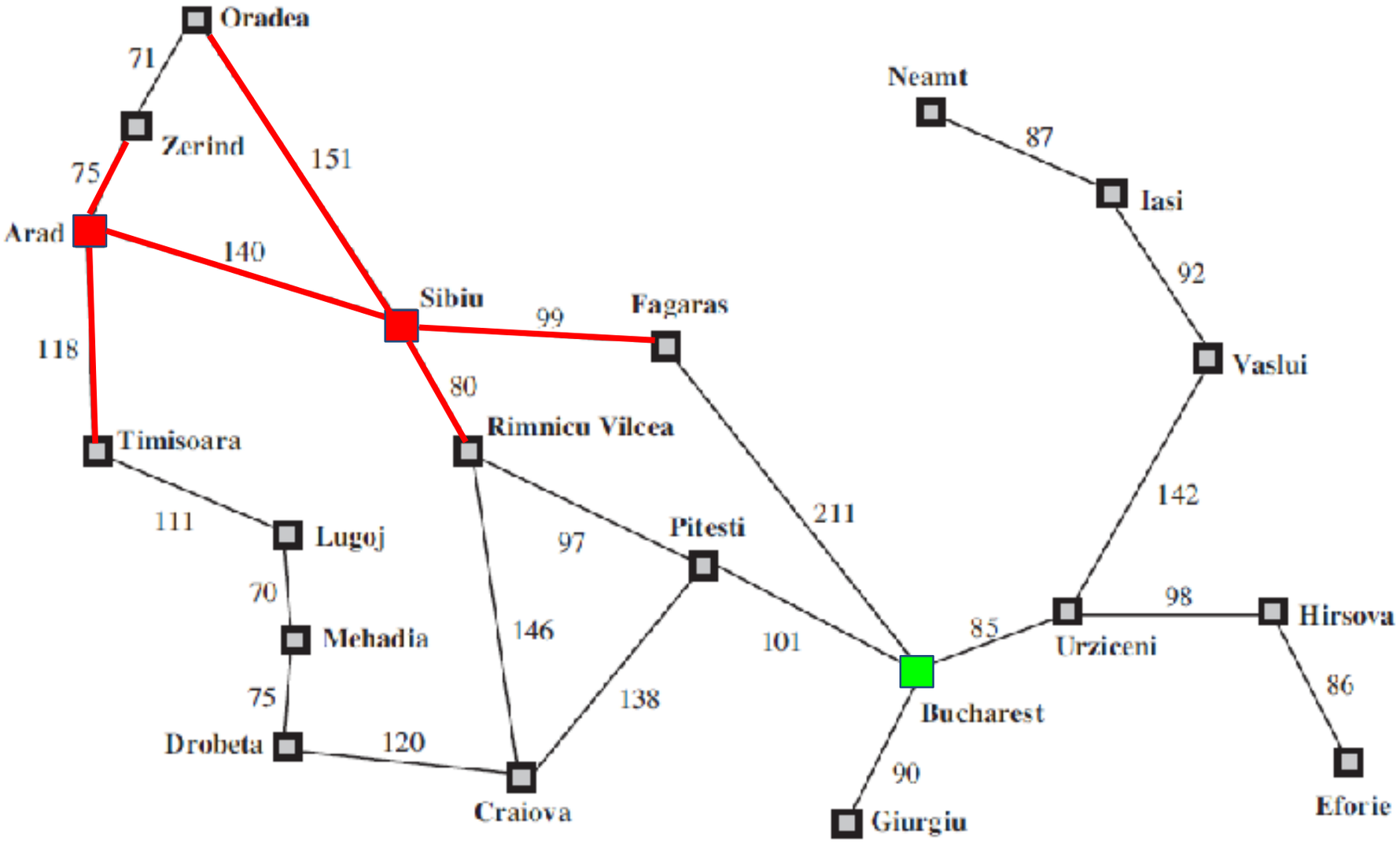

(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193



...

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

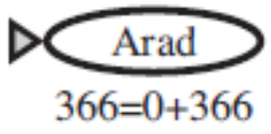

(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193



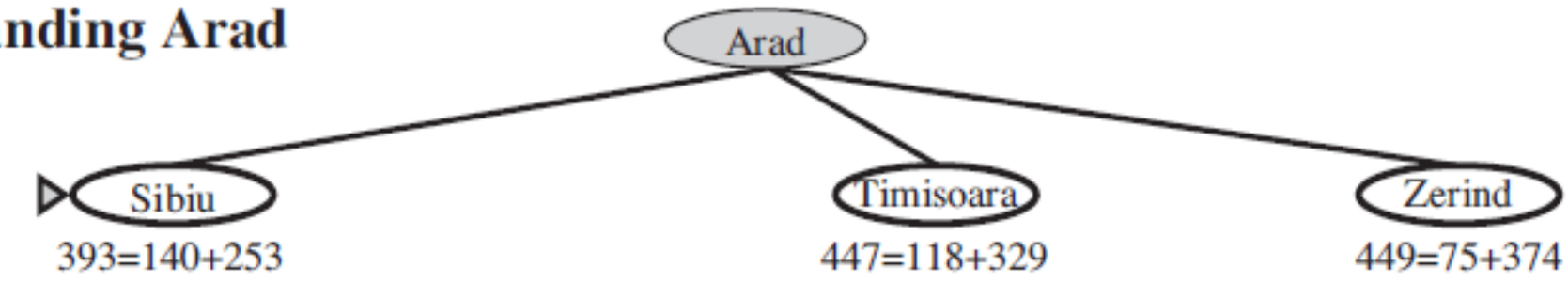
...

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

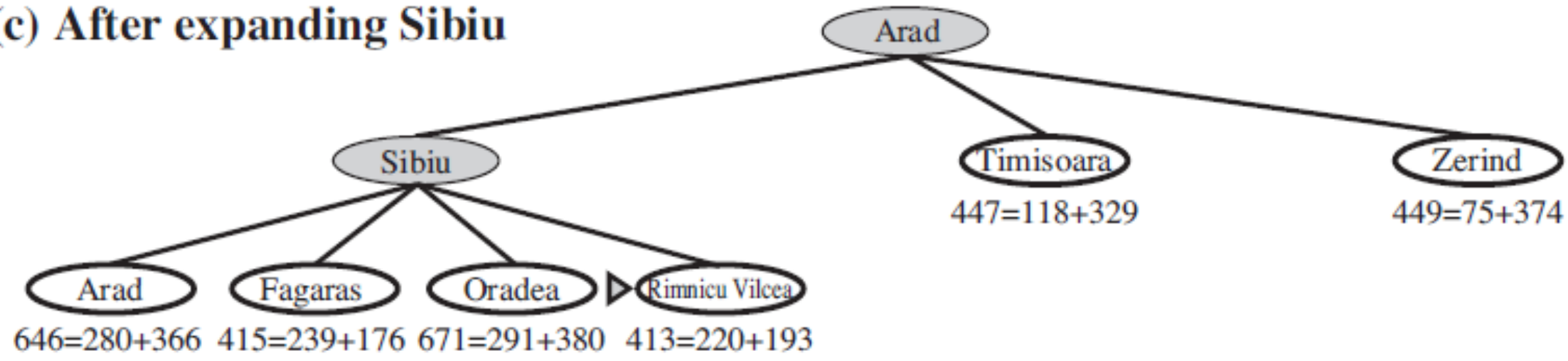
(a) The initial state

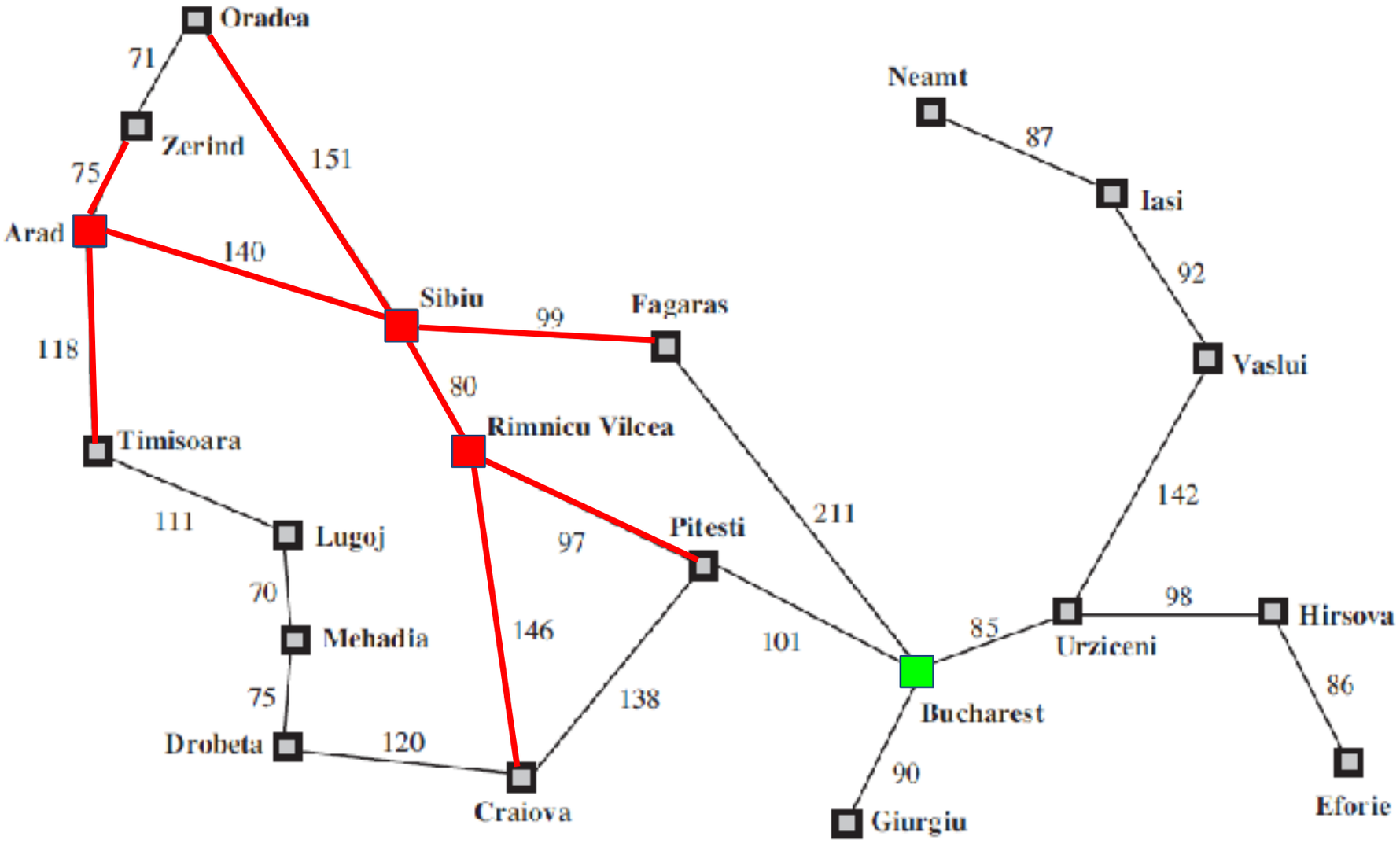


(b) After expanding Arad



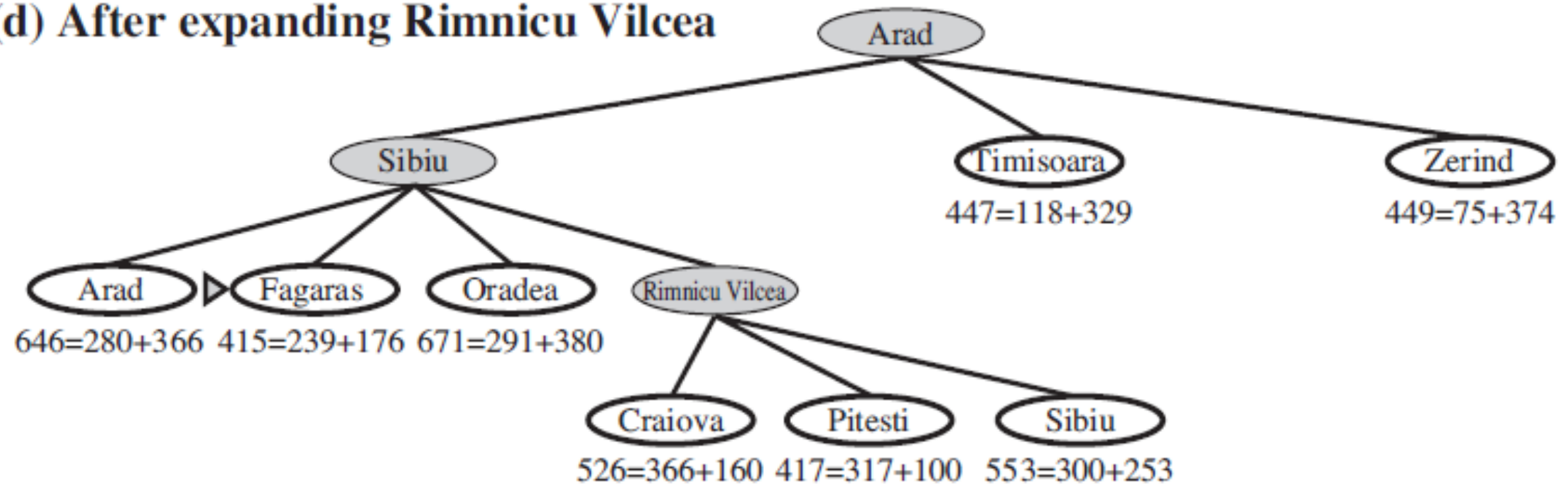
(c) After expanding Sibiu

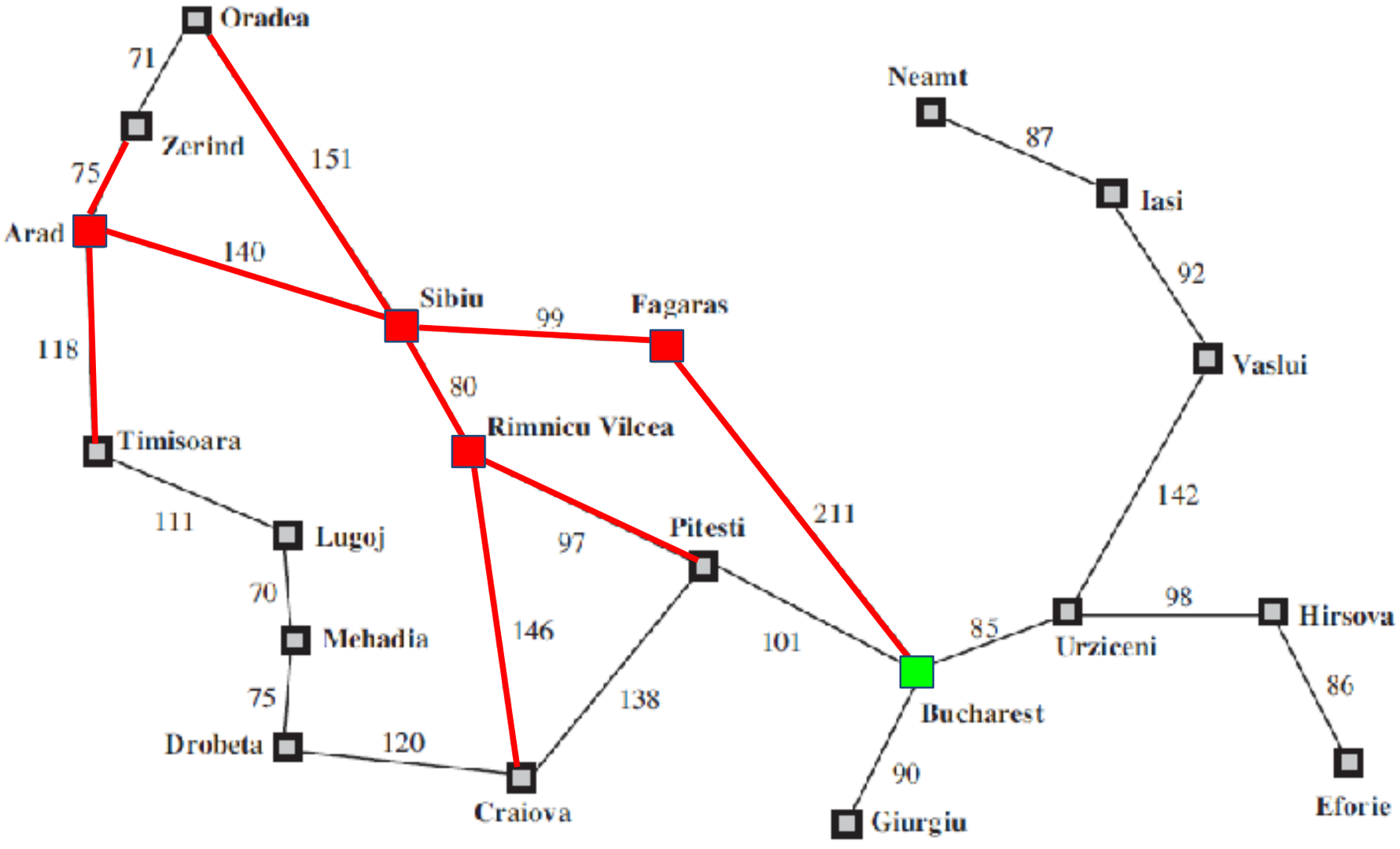




Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

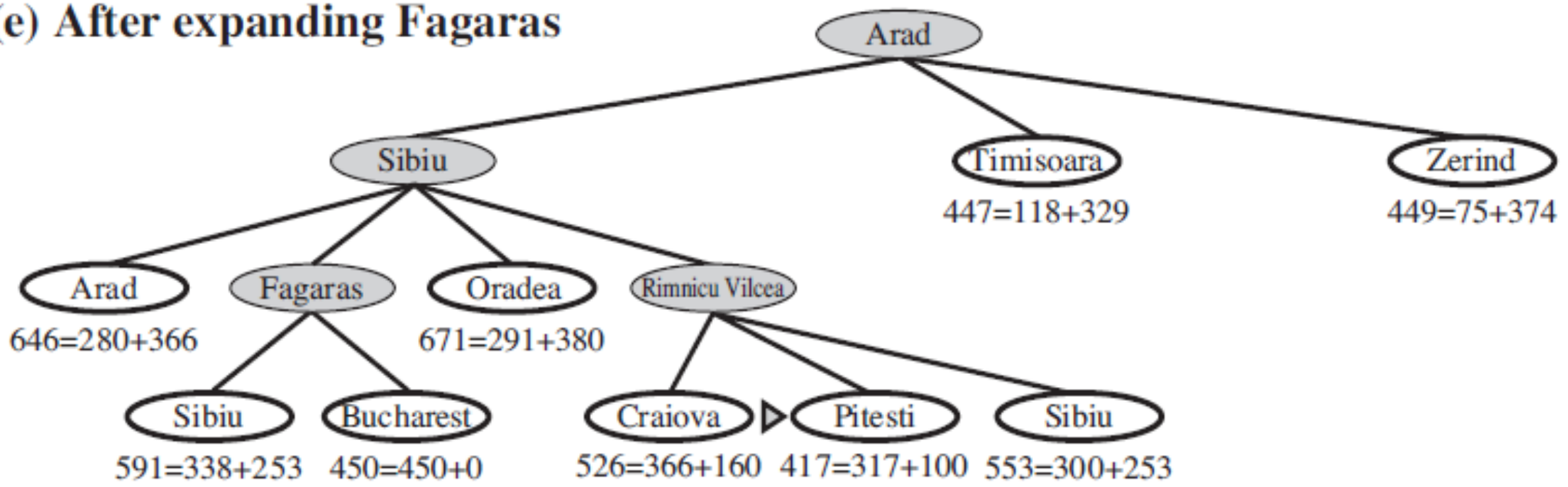
(d) After expanding Rimnicu Vilcea

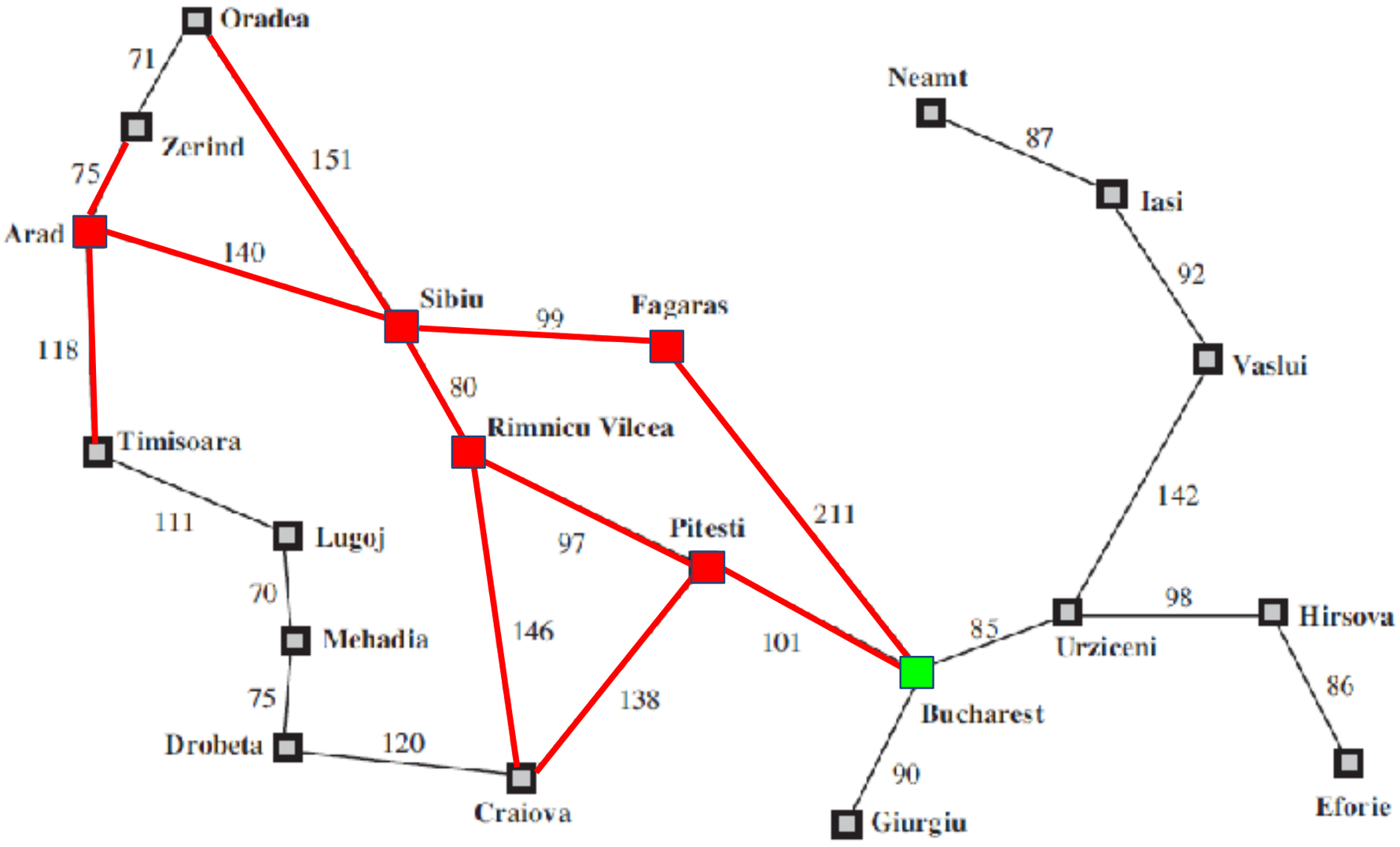




Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

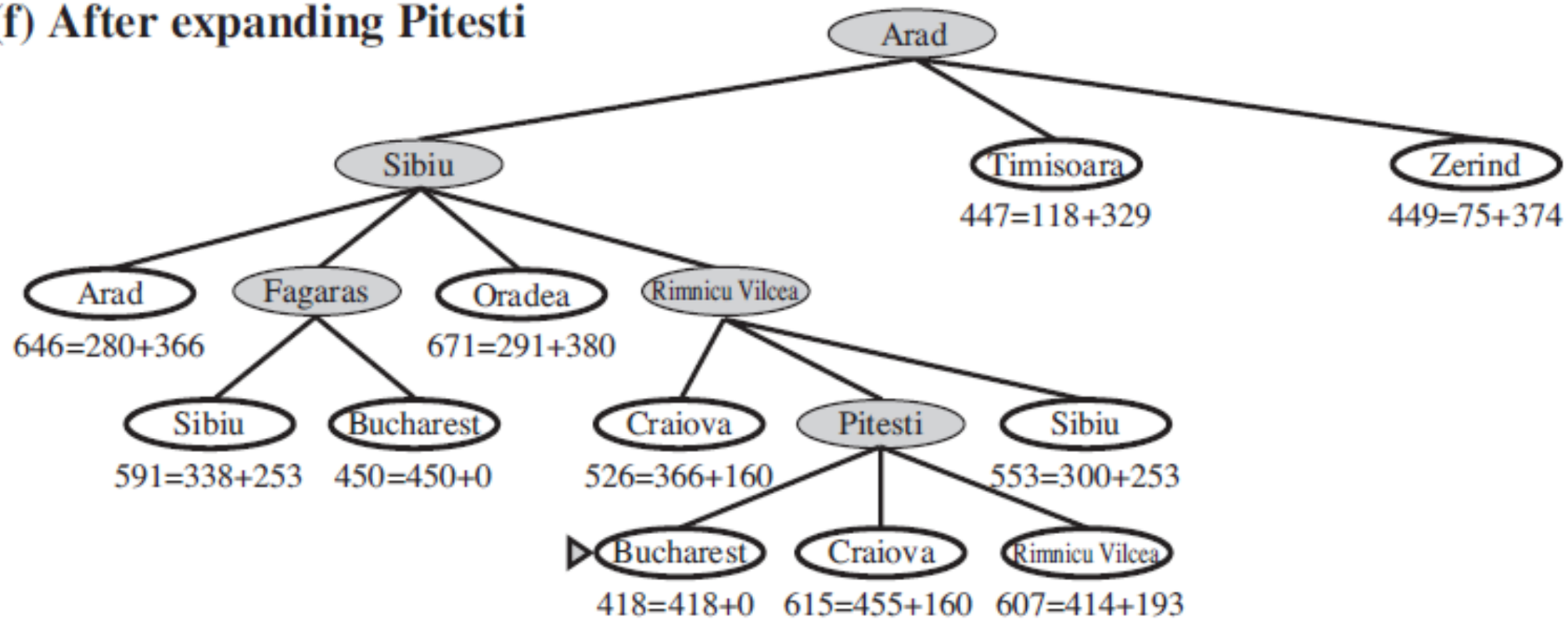
(e) After expanding Fagaras



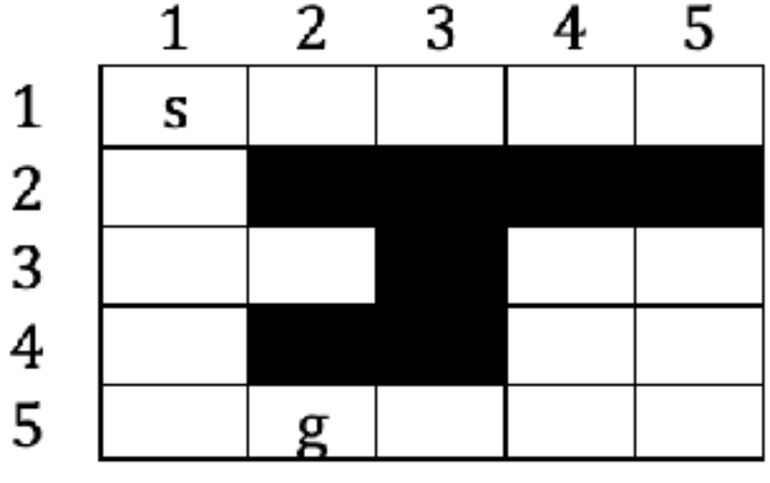


Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

(f) After expanding Pitesti



A* QUIZ



Starting state = (1,1)
Goal state = (2,5)

g. Heuristic function ที่จะเลือกใช้แก้ปัญหาแผนที่นี้ จะเลือกใช้อะไร และสิ่งที่เลือกใช้ คืออะไร

h. Evaluation function สำหรับการทำ A* Search บนแผนที่นี้ คืออะไร แต่ละองค์ประกอบของ evaluation function หมายความว่าอะไรบ้าง

i. จงวาด tree ที่เกิดจากการทำ path planning โดยใช้ A* Algorithm ให้ใส่ตัวเลขที่แต่ละกิ่ง(branch) เพื่อแสดงลำดับการแตกกิ่ง

THE ALGORITHM

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

THE ALGORITHM

- Another way to implement A*

  *https://en.wikipedia.org/wiki/A*_search_algorithm*

ADVANCED TOPICS

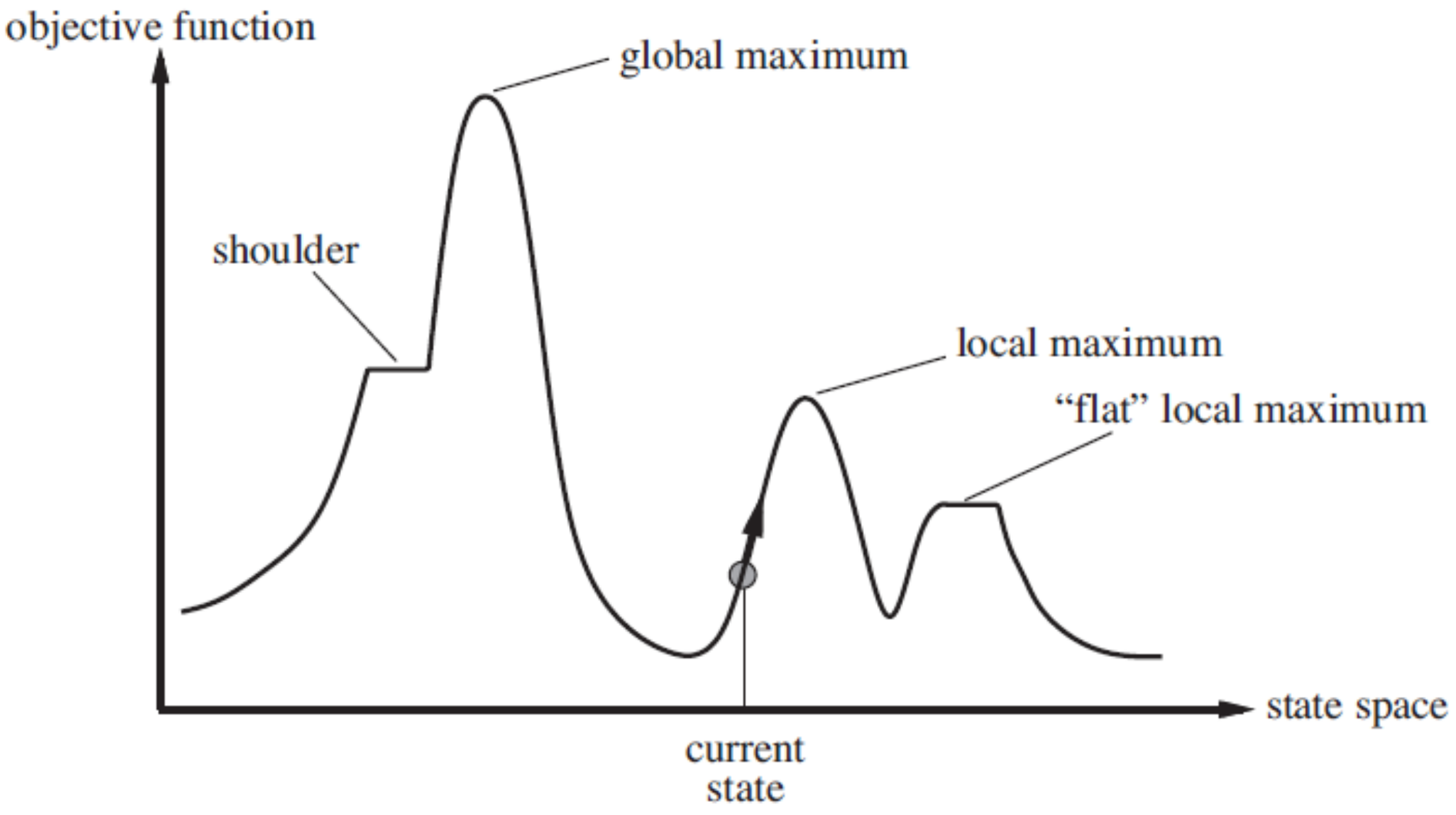
Nice to know, but not a must-know
(We will hold this lecture later)

LOCAL SEARCH & OPTIMIZATION PROBLEM

LOCAL SEARCH

- Key Idea:
  - Have only one node/robot. Try to find the goal.

- Definition:
  - The algorithms operate using a single current node and generally move only to neighbors of that node.


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

HILL CLIMBING SEARCH

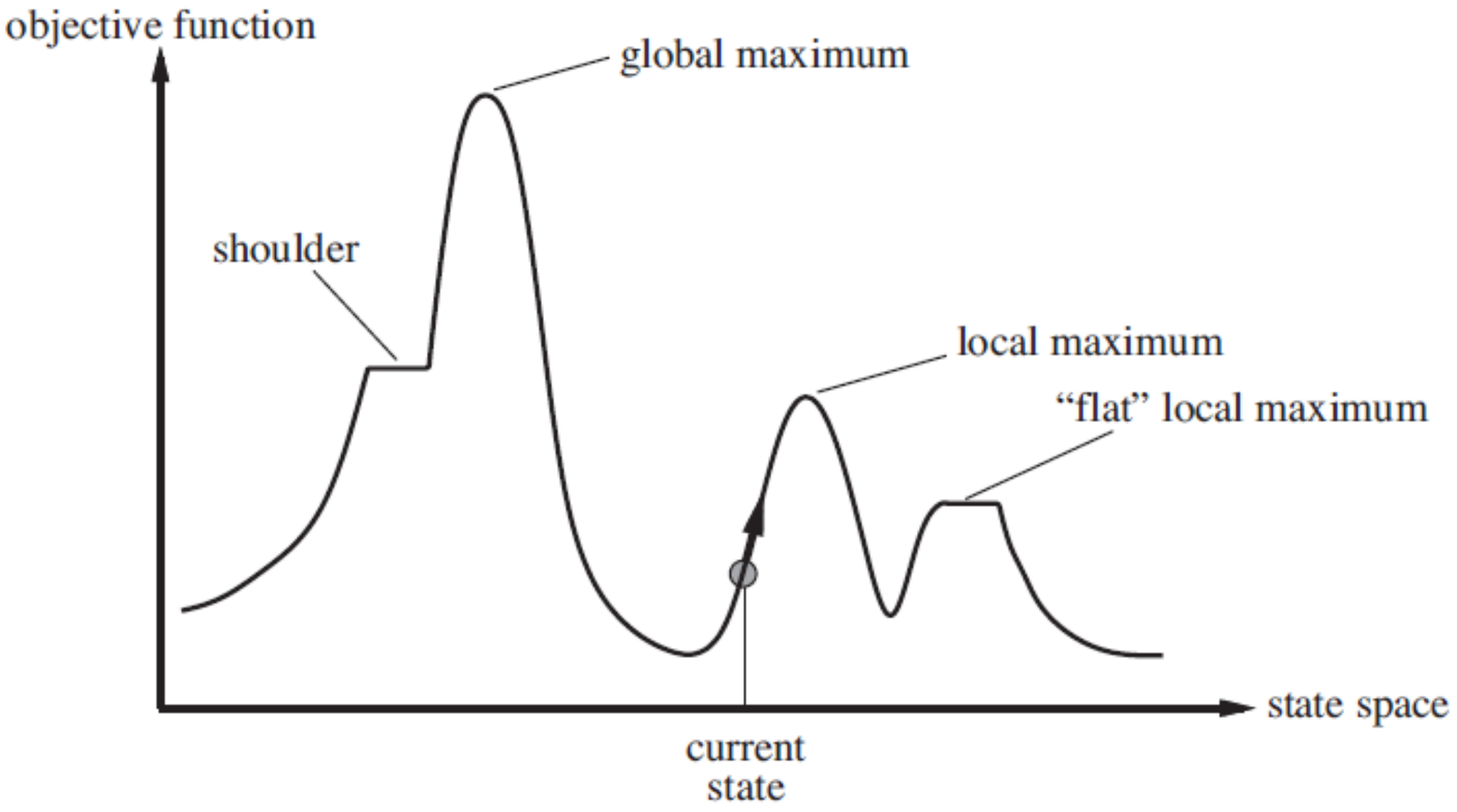
Local Search and Optimization Problem

HILL CLIMBING SEARCH

- Key Idea:
  - Just like climbing the hill, trying to reach the top
  - Always go uphill (higher)

- Definition:
  - The algorithm 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 neighbor has a higher value.

EXAMPLE

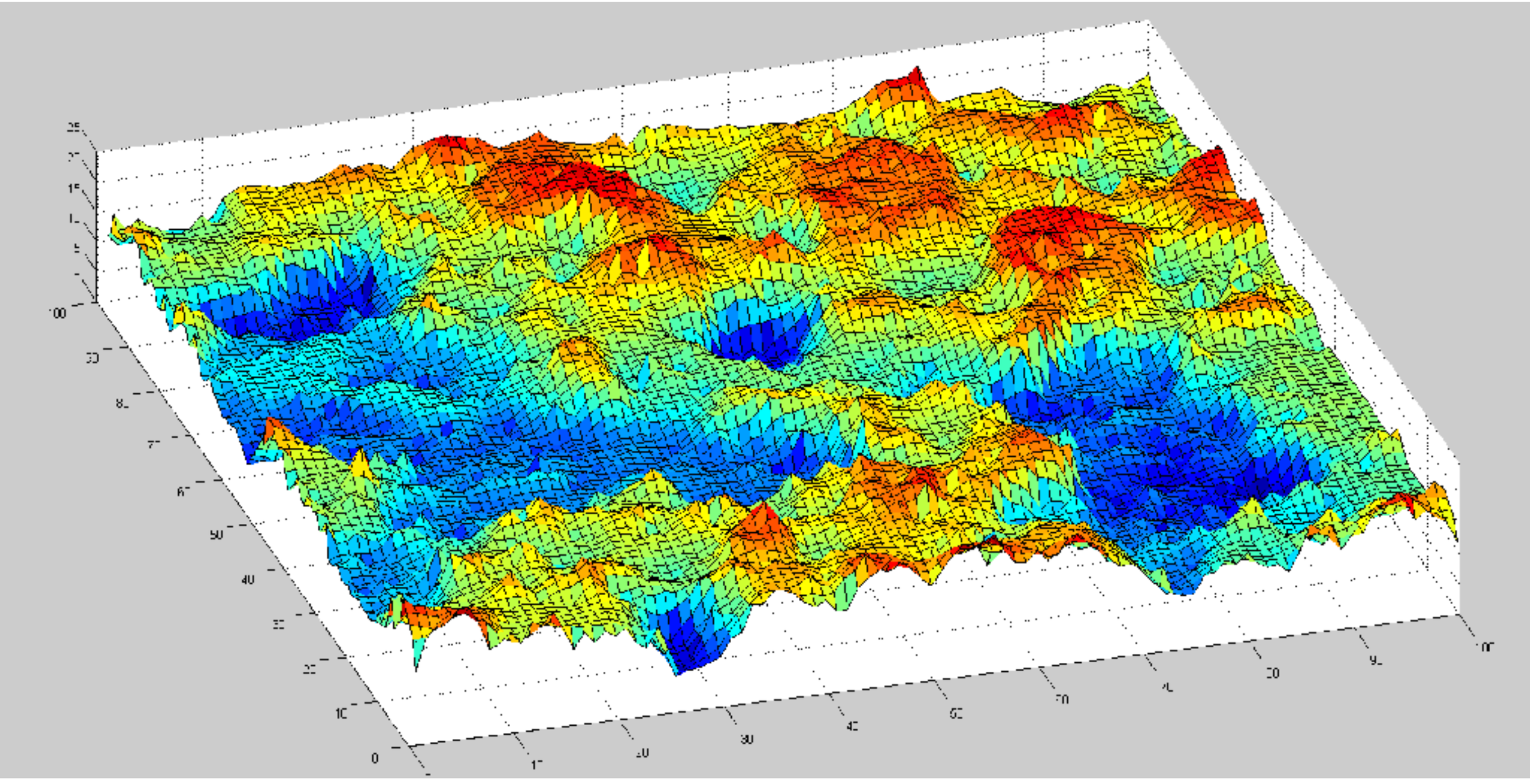

objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

HILL CLIMBING: THE ALGORITHM

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

$current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
loop do
$neighbor \leftarrow$ a highest-valued successor of $current$
if neighbor.VALUE $\leq$ current.VALUE **then return** $current$.STATE
$current \leftarrow neighbor$

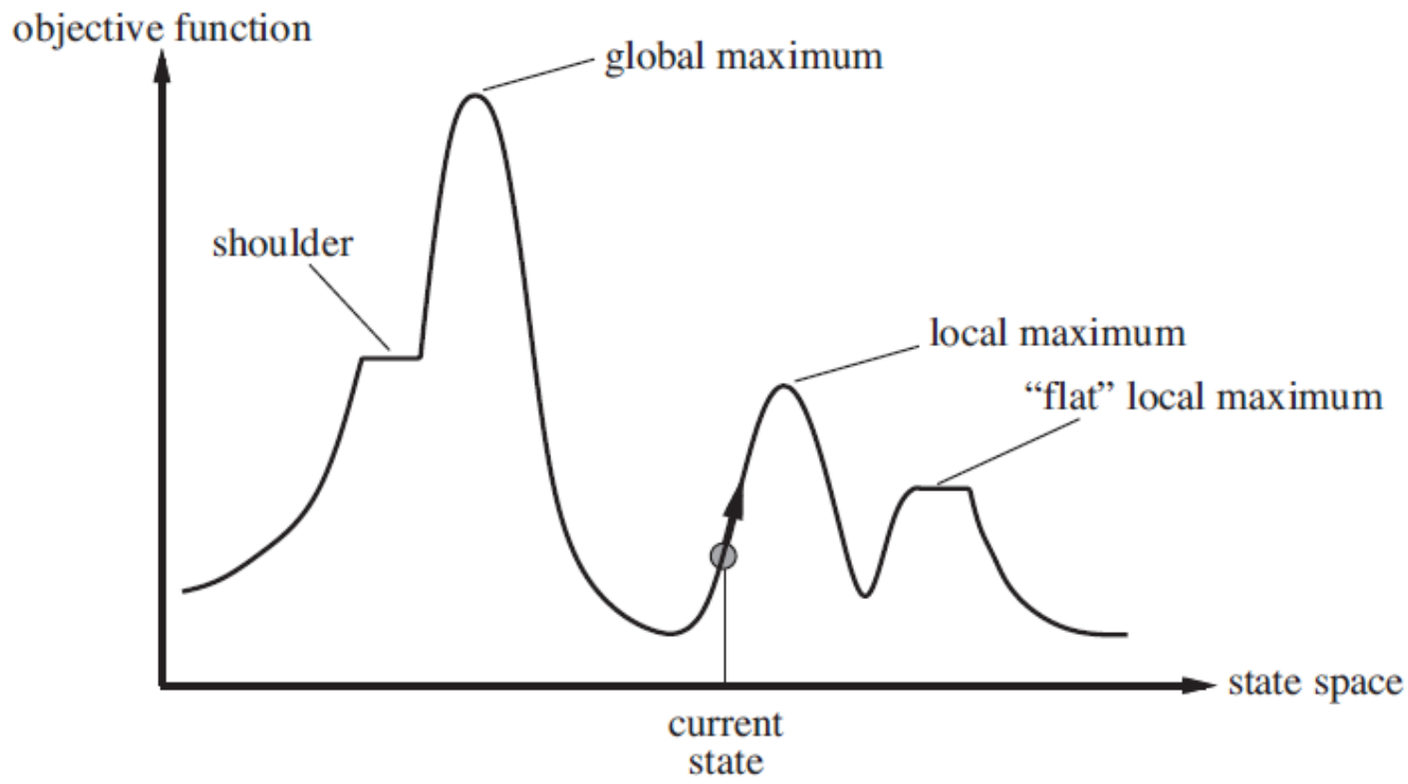
POTENTIAL PROBLEMS



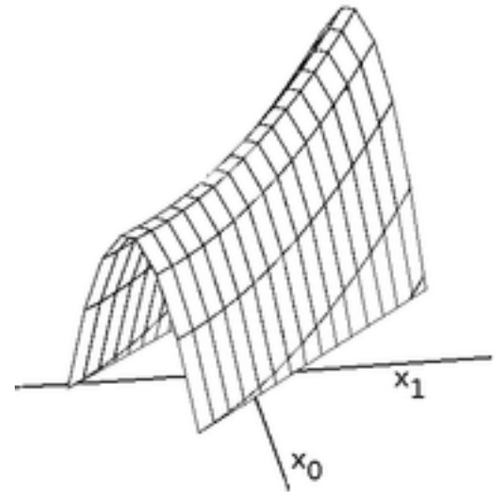
➤ Retrieved from: http://i.imgur.com/89KoR.png

POTENTIAL PROBLEMS

- Local Maxima
- Plateau
  - A flat plain



- Ridge
  - Any actions leads to lower ground, but combined actions lead to higher ground.



SOLVING PROBLEMS

- How to solve local minima, plateau, and ridge problems?

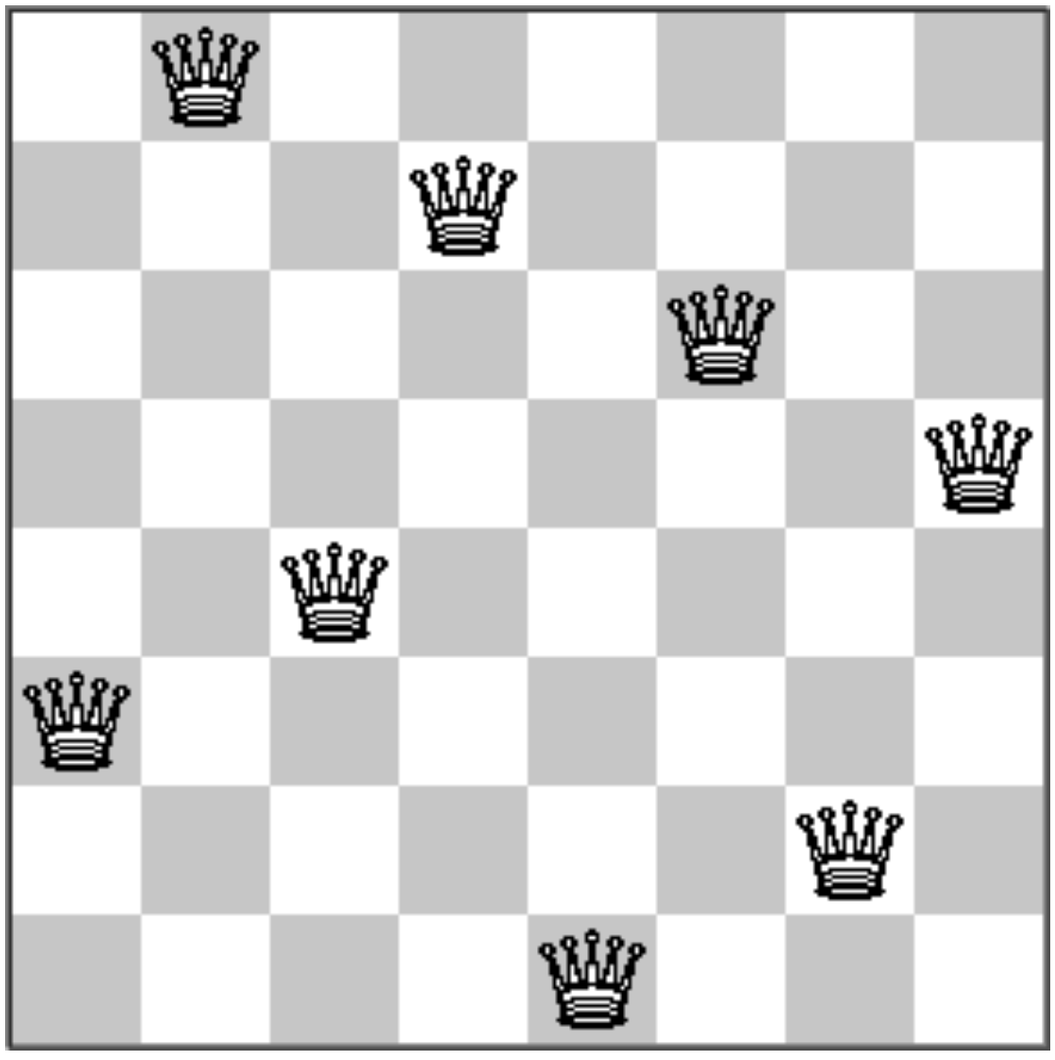
SOLVING PROBLEMS

- How to solve local minima, plateau, and ridge problems?
  - Random starting points
  - Backtracking
    - If we encounter the problem, just roll back some actions, and try a different one.
- Simulated Annealing
  - Do not always choose the best actions. Probabilistically allow some bad actions and go with it. It may lead to a better solution.

GENETIC ALGORITHM

Local Search and Optimization Problem

INTRODUCING 8 QUEENS PROBLEM



Retrieved from: http://letstalkdata.com/2013/12/n-queens-part-1-steepest-hill-climbing/

GENETIC ALGORITHM

Key Idea:

- Crossover and Mutation in DNA
- Natural Selection

Definition:

- Genetic Algorithm is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.

GENETIC ALGORITHM: DEMO

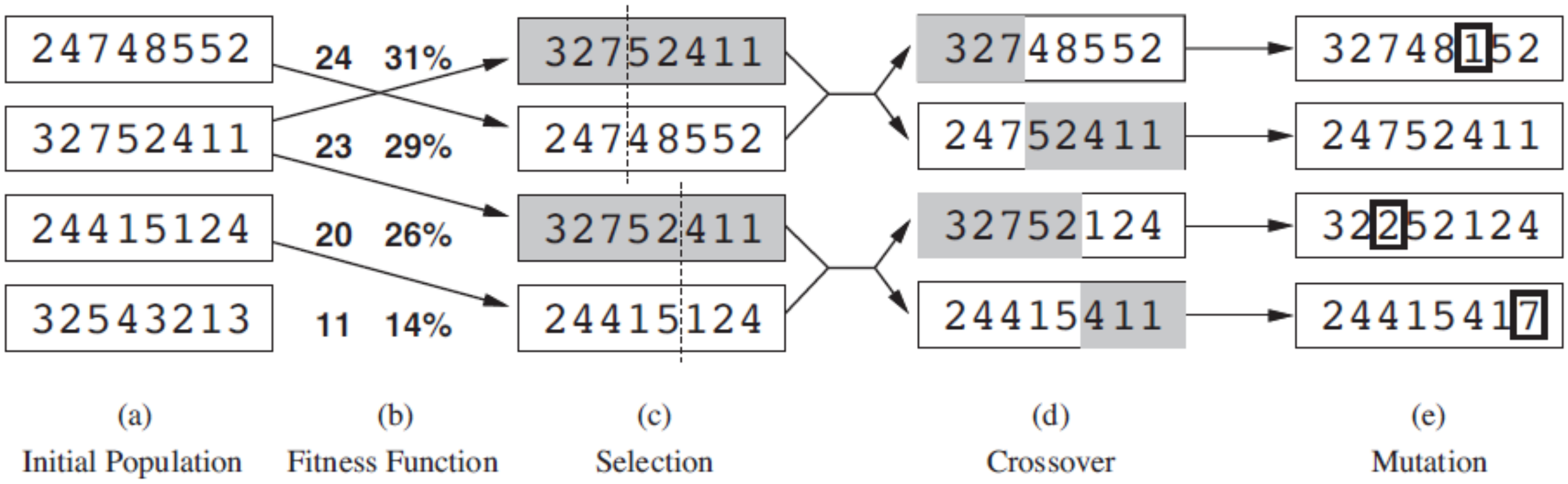
Genetic Cars

http://rednuht.org/genetic_cars_2/

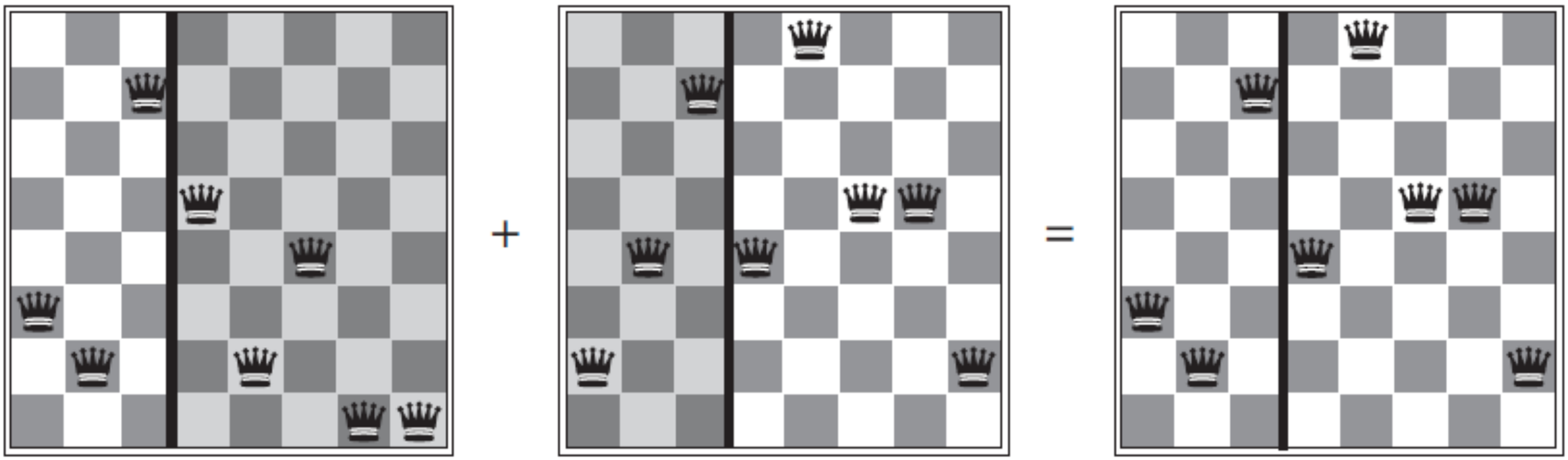
6 Species trying to reach the goal

https://www.youtube.com/watch?v=2Q9bF8Ofyho&list=PLpJFWWZXdN_qaI1SR6-7j6-5MyxhRqLQ0&index=2

GA: EXAMPLE


24748552
32752411
24415124
32543213
24
31%
23
29%
20
26%
11
14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Crossover
(e)
Mutation

GA: EXAMPLE



```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
      new_population ← empty set
      for i = 1 to SIZE(population) do
          x ← RANDOM-SELECTION(population, FITNESS-FN)
          y ← RANDOM-SELECTION(population, FITNESS-FN)
          child ← REPRODUCE(x, y)
          if (small random probability) then child ← MUTATE(child)
          add child to new population
      population ← new population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

SEARCHING WITH NO OBSERVATION

WHAT IF...

The agent doesn't know where it is in the world.

- The agent knows only what it can do.
    - Know all action it can do.
    - Know all possible states.
- How to reach the goal state?

SEARCHING WITH NO OBSERVATION

Key idea:

- The robot doesn't know anything about positioning or environments (sensorless).
- It tries to reach the goal state.

Definition:

- When the agent's percepts provide no information at all, we have what is called a sensorless problem.

INTRODUCING VACUUM WORLD PROBLEM

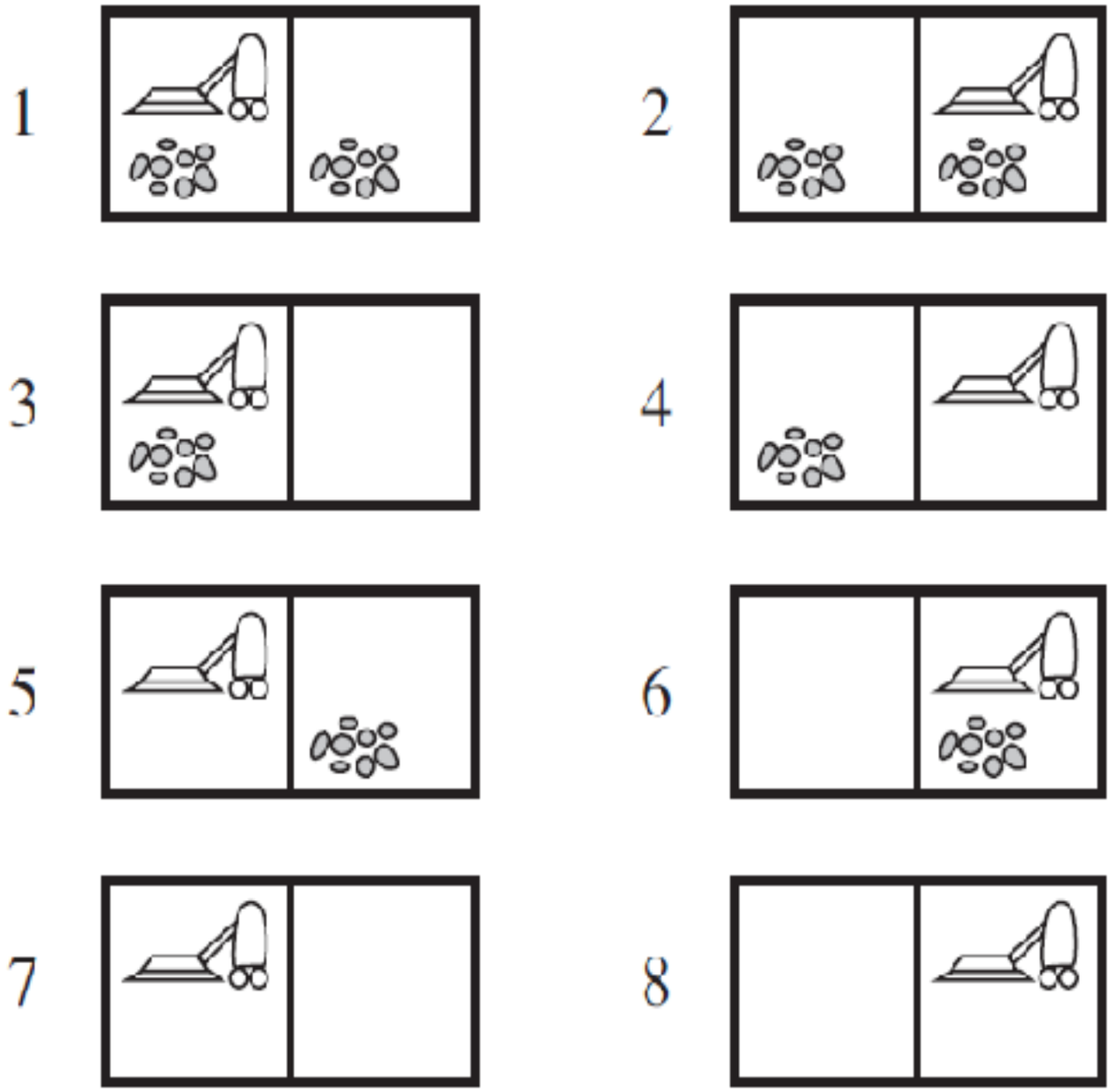
1 Robot

2 Rooms

- Can be Clean/Dirty

3 Actions

- Left
- Right
- Suck



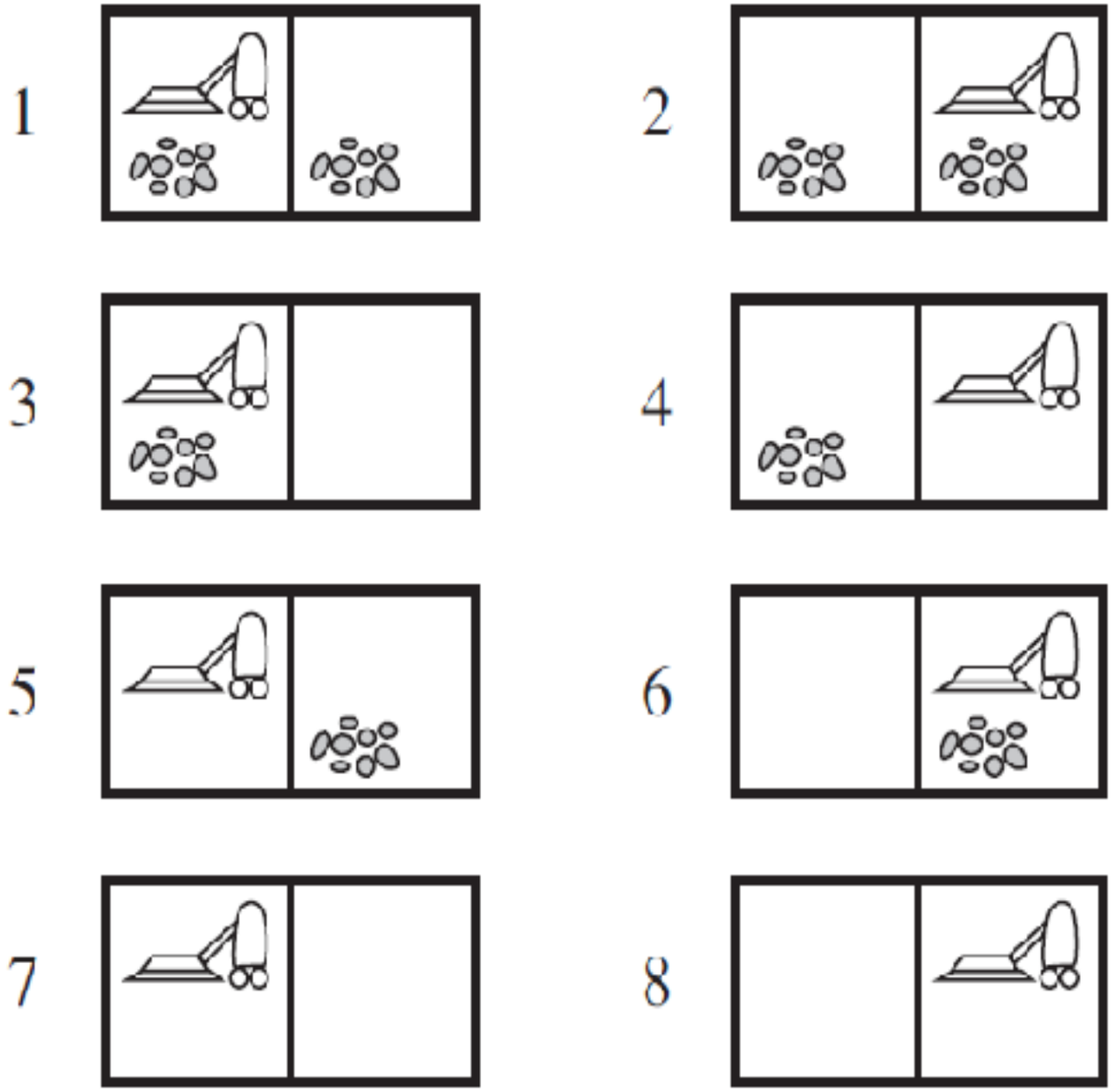
INTRODUCING VACUUM WORLD PROBLEM

Goal: clean both room.

Question:

- What actions does the robot have to take?

Group up, and try to derive the algorithm.



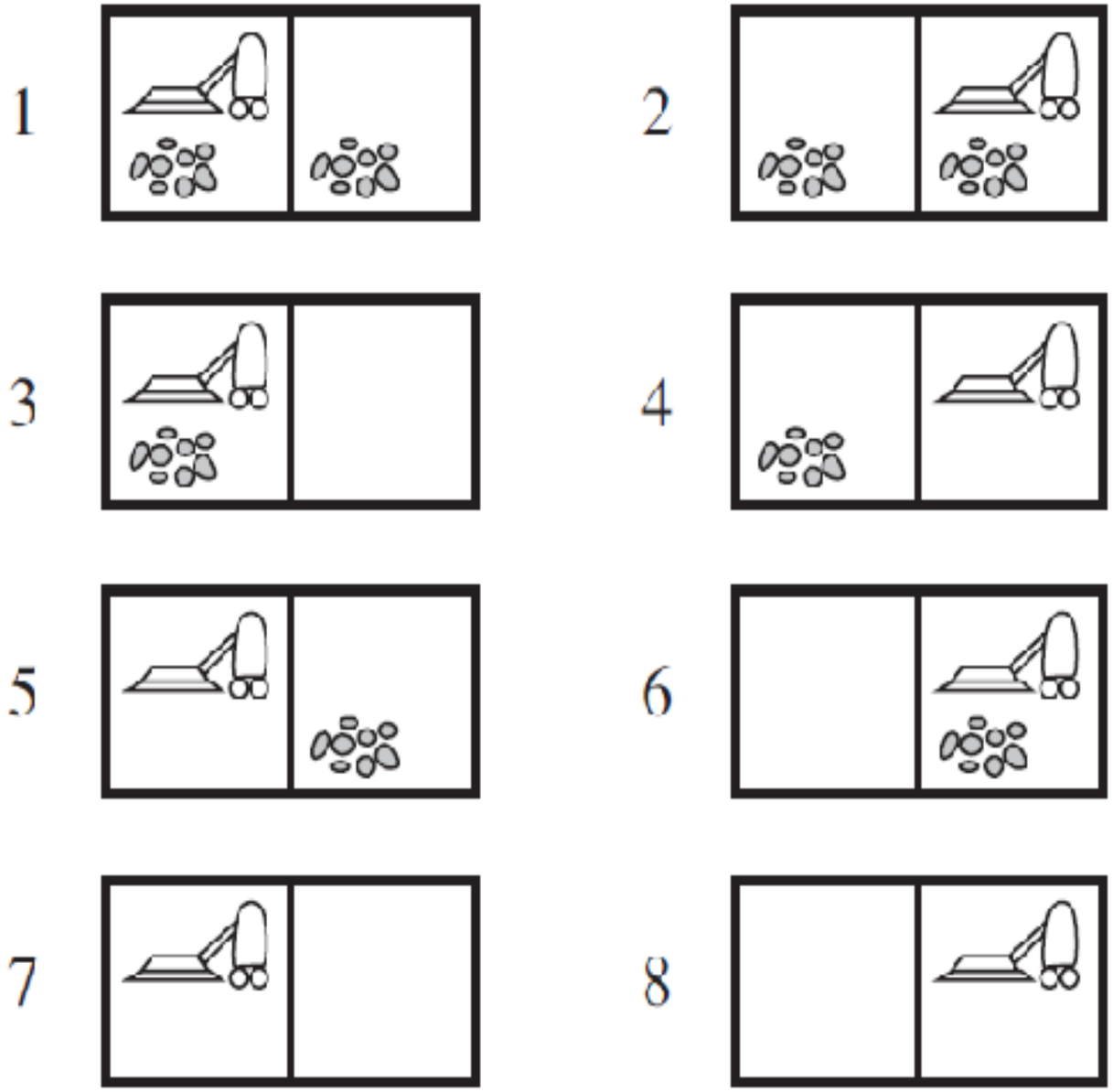
HOW TO SOLVE SENSORLESS PROBLEM

Find Belief States

- Every possible set of states, even if some states are unreachable from the initial state

Vacuum World

- There are 2 rooms. Each room is either clean or dirty
- Total belief states = 2*2*2 = 8



HOW TO SOLVE SENSORLESS PROBLEM

We have all possible states.

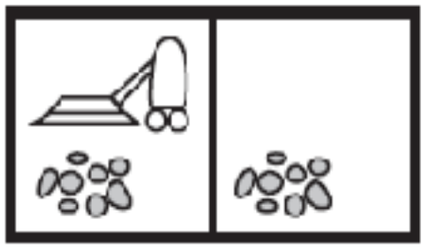
Next step:

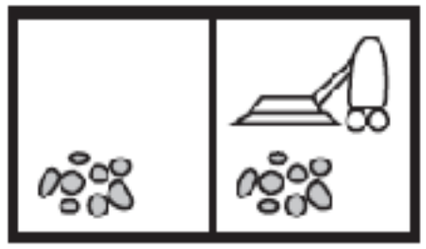
- Find possible action for each state.
    - Legal state: an agent can perform that action.
    - Illegal state: an agent cannot perform that action.
- Assumed that illegal state cannot break the world, we union all the possible actions.
- Assumed that illegal state can break the world, we intersect all the possible actions.

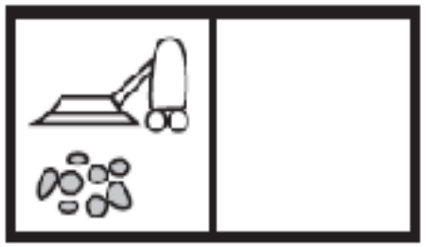
HOW TO SOLVE SENSORLESS PROBLEM

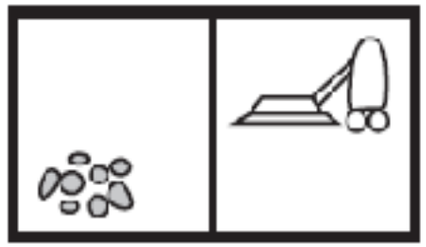
Original Action: Left, Right, Suck

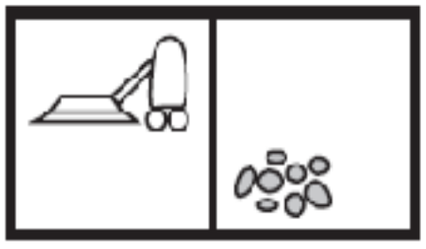
Assume that illegal actions cannot break the world, what are the actions?

1 

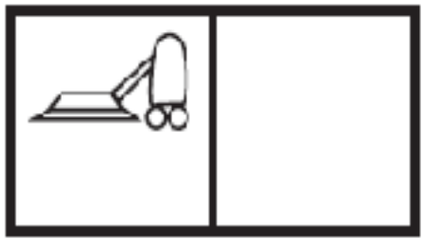
2 

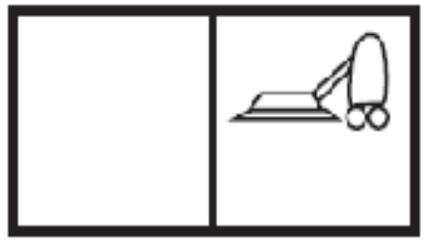
3 

4 

5 

6 

7 

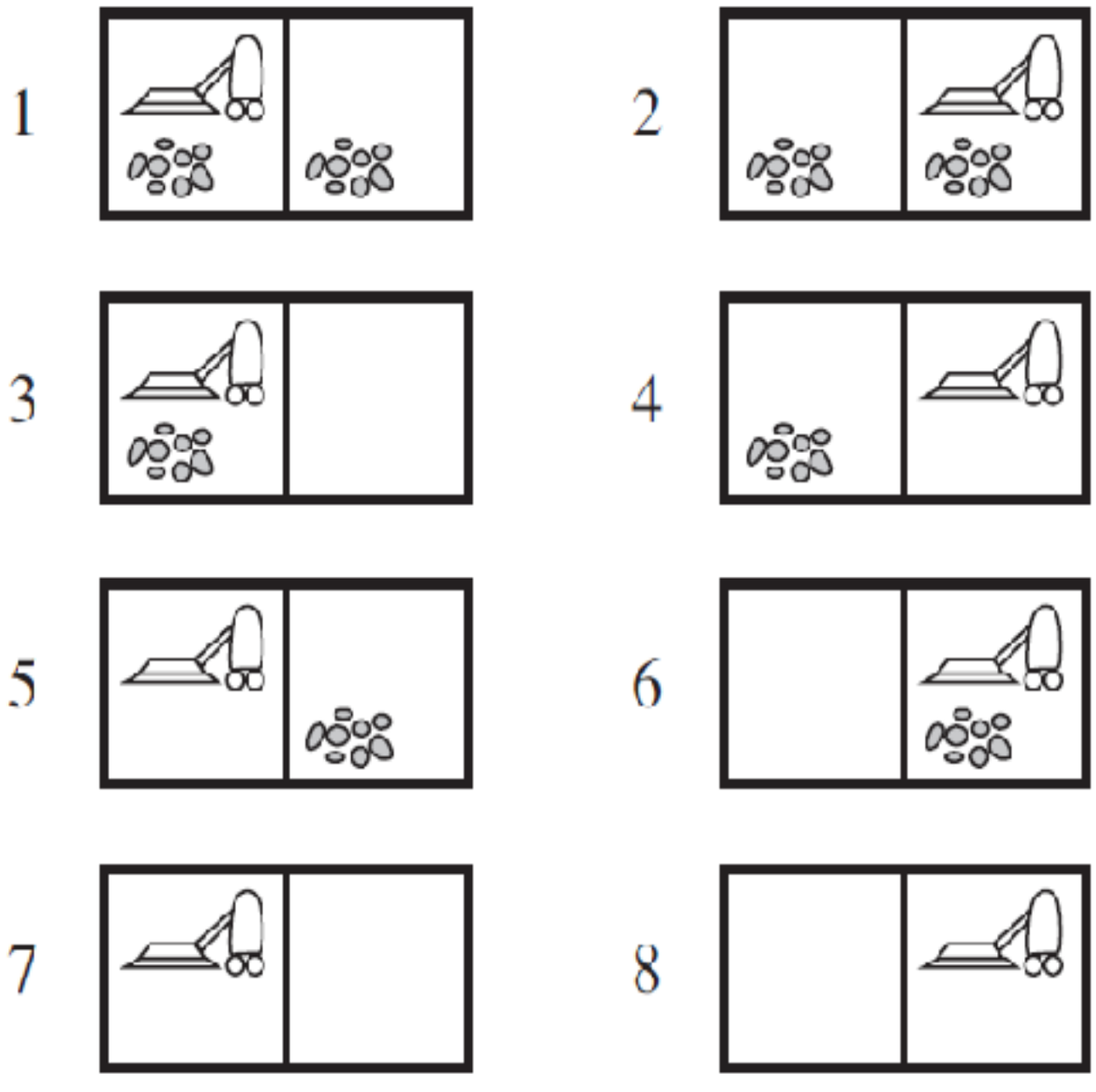
8 

HOW TO SOLVE SENSORLESS PROBLEM

Original Action: Left, Right, Suck

Assume that illegal actions cannot break the world, what are the actions?

- Left
- Right
- Suck

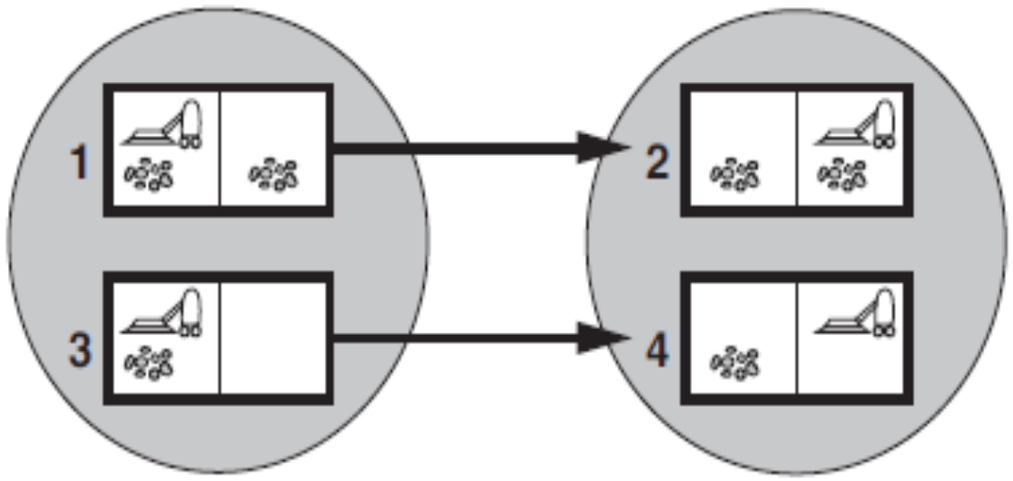


HOW TO SOLVE SENSORLESS PROBLEM

We have states, and actions.

Next step:

- Transition Model
  - Perform actions on states

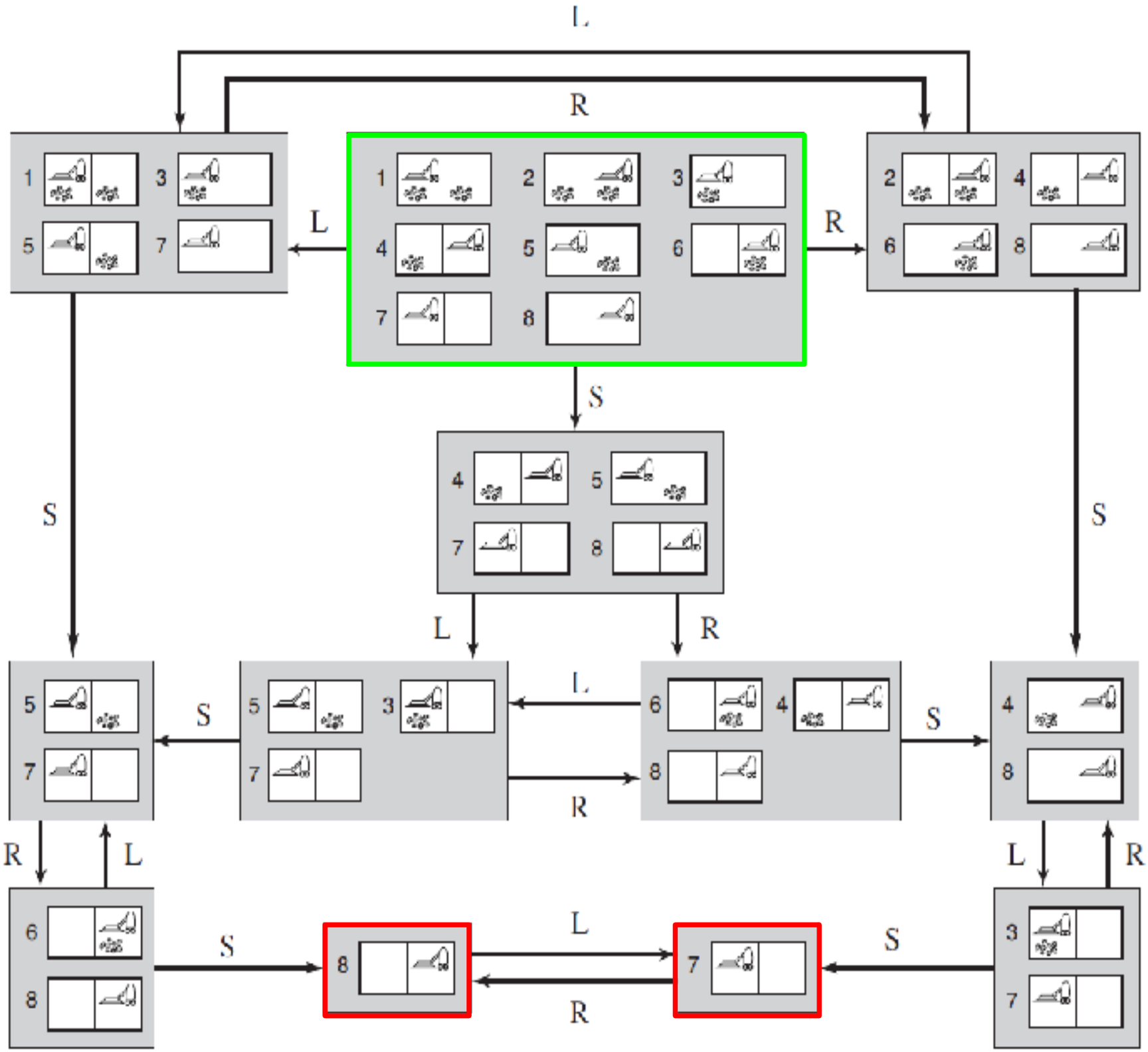


HOW TO SOLVE SENSORLESS PROBLEM

Finally:

- Goal Test
    - Test if that state is the goal state or not

- Path Cost
    - Calculate cost of the path taken.


L
R
1
3
5
7
L
1
2
3
4
5
6
7
8
R
2
4
6
8
S
S
4
5
7
8
L
R
5
7
S
5
3
7
L
R
6
4
8
S
4
8
R
L
L
R
6
8
S
8
L
R
7
S
3
7

SEARCHING WITH PARTIAL OBSERVATION

SEARCHING WITH PARTIAL OBSERVATION

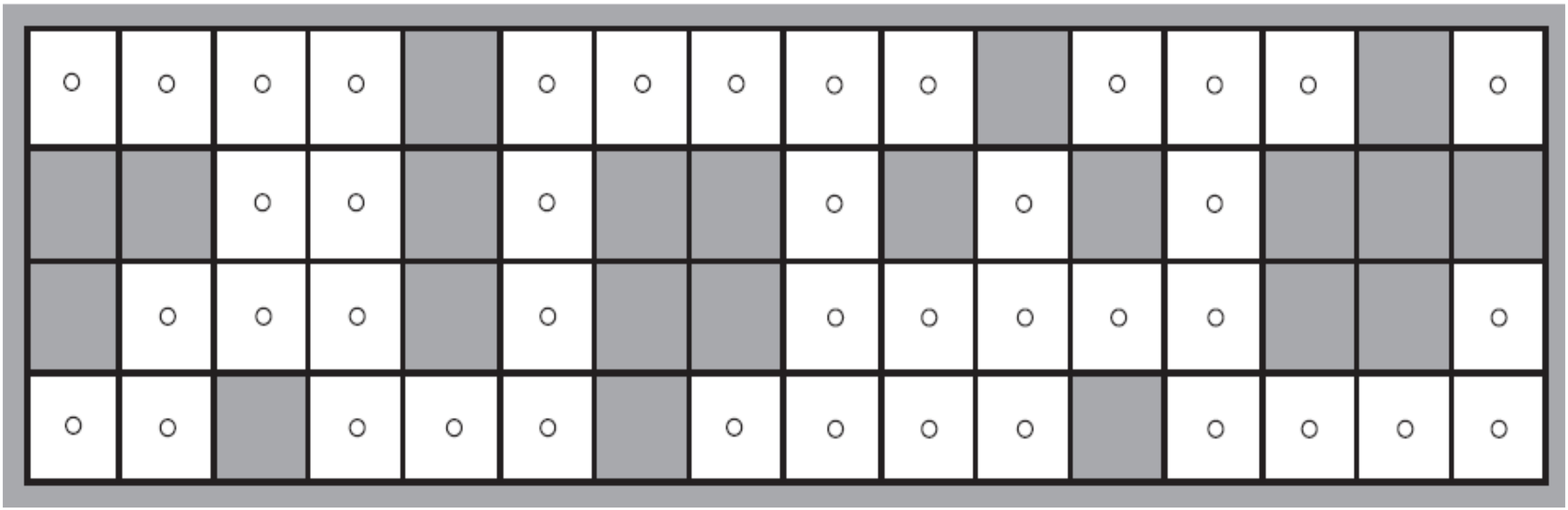
Key Idea:

- The agent observes the current state.
- The agent updates the knowledge base.
- The agent performs actions.
- The agent checks for the goal state.
- Loop back to the first state, if goal state is not reached.

INTRODUCING ROBOT LOCALIZATION PROBLEM

We don’t know where the robot is

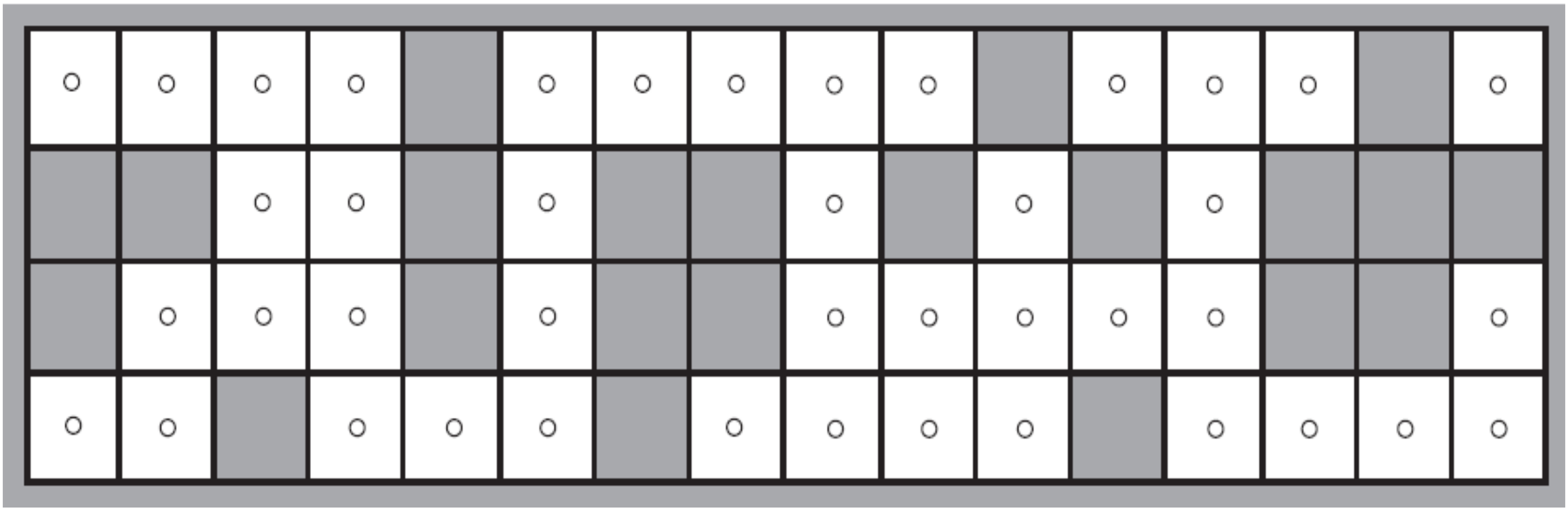
BUT the robot can report its environment !!



ROBOT LOCALIZATION PROBLEM

The robot just sent a signal NSW.

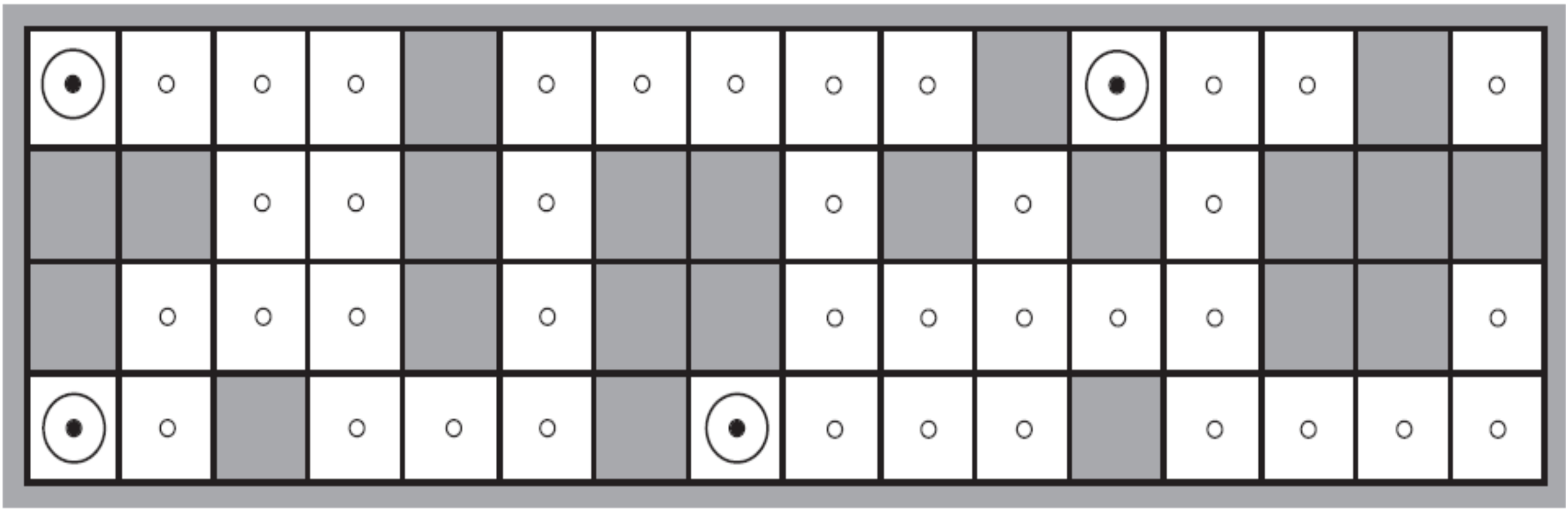
This means that the robot cannot move North, South, and West.



ROBOT LOCALIZATION PROBLEM

The robot just sent a signal NSW.

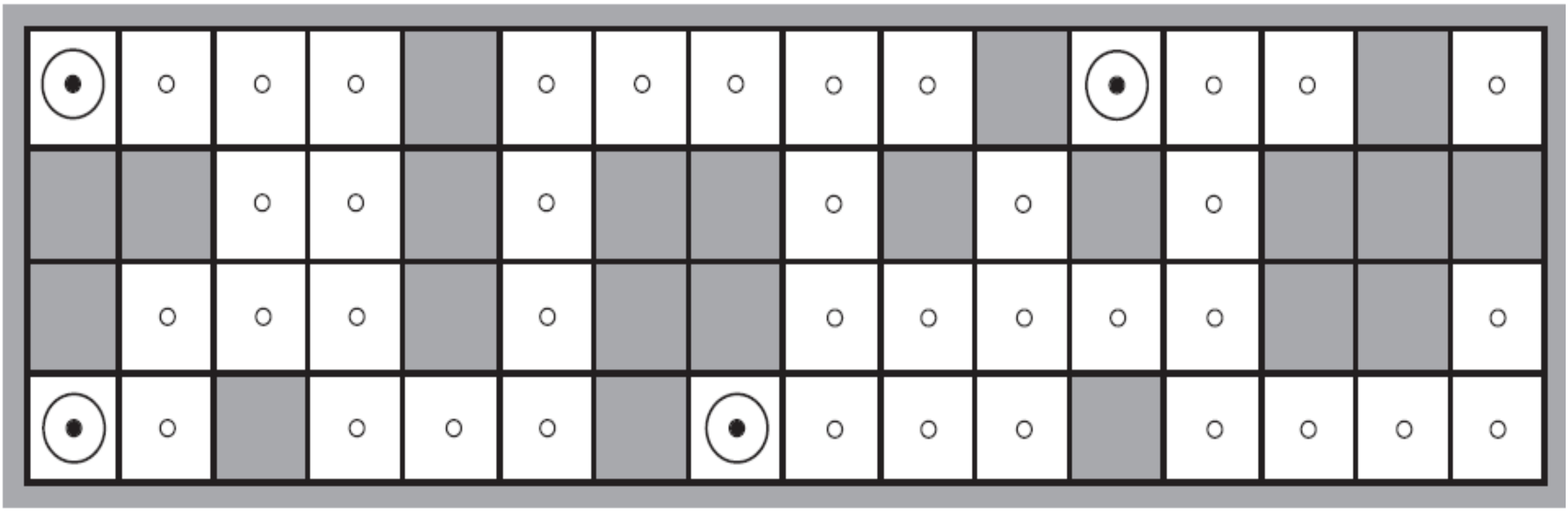
Let's update the map.



ROBOT LOCALIZATION PROBLEM

From the signal NSW, there are 4 possible locations.

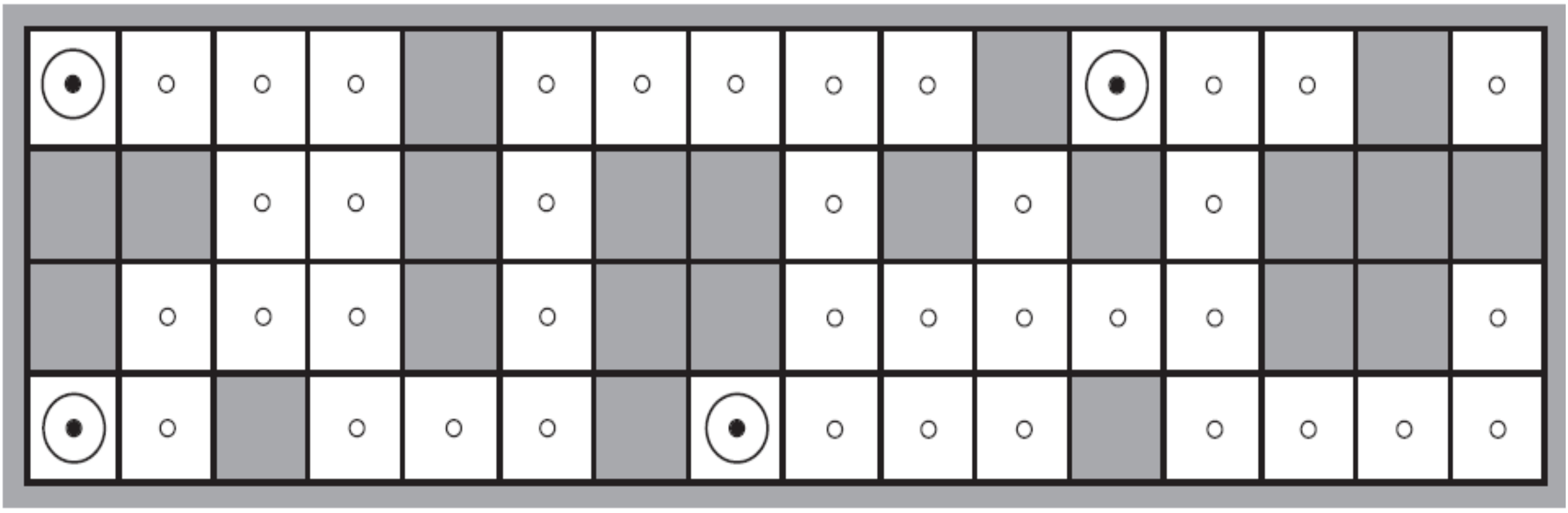
Let's move the robot right.



ROBOT LOCALIZATION PROBLEM

The robot just sent another signal NS.

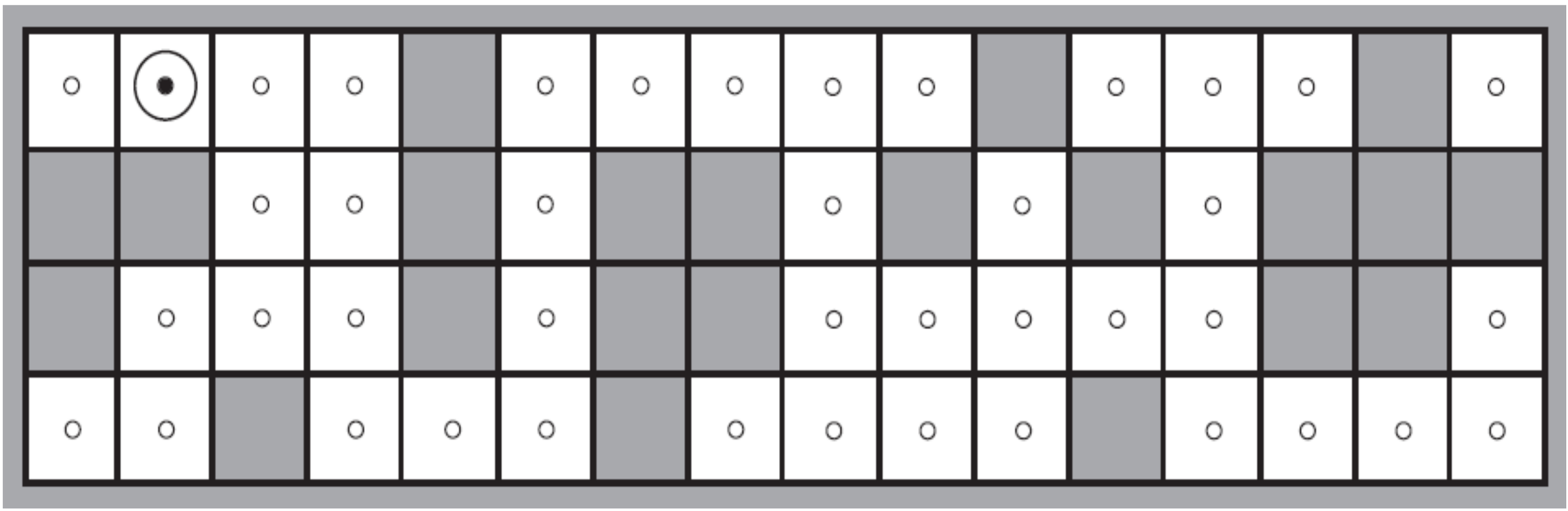
It means that the robot cannot move North and South.



ROBOT LOCALIZATION PROBLEM

The robot just sent a signal NS.

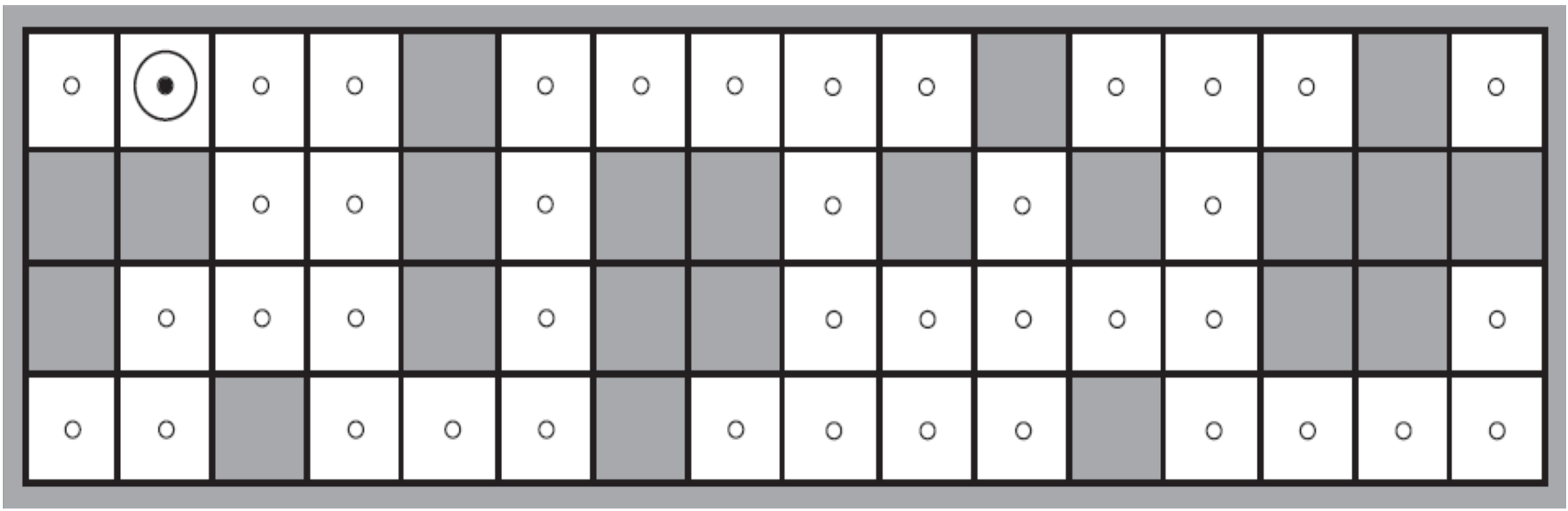
Let's update the map again.



ROBOT LOCALIZATION PROBLEM

We can also call the grid below an occupancy grid.

A grid of binary random variable in each cell. Each represents a presence of an obstacle.



ROBOT LOCALIZATION PROBLEM

Extending the algorithm we learnt so far, we can perform a Particle Filter.

We won't go into the mathematical details of Particle Filter.

However let's learn the concept:

https://www.youtube.com/watch?v=aUkBa1zMKv4&t=247s

MINIMAX

Game Search

MINIMAX

- Let's look at the animation of minimax first.

- https://www.youtube.com/watch?v=KU9Ch59-4vw&t=63s

MINIMAX

Key Idea:

- Playing games between 2 players: Checkers, Chess, …
- Maximize outcomes of our plays, and minimize outcomes of opponent's plays

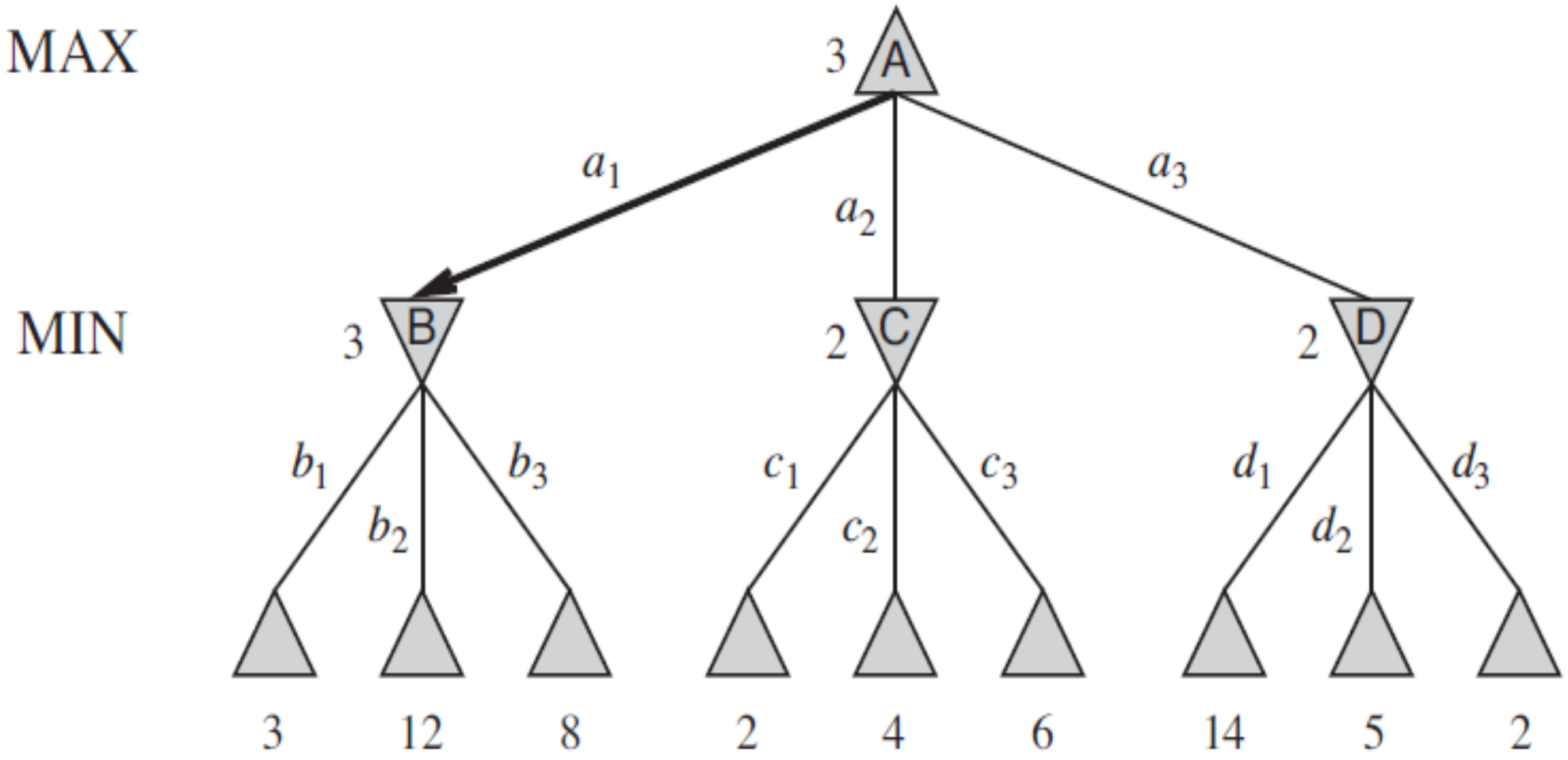
Definition:

- $\text{Minimax}(s) = \text{Utility}(s)$ if Terminal-Test(s)

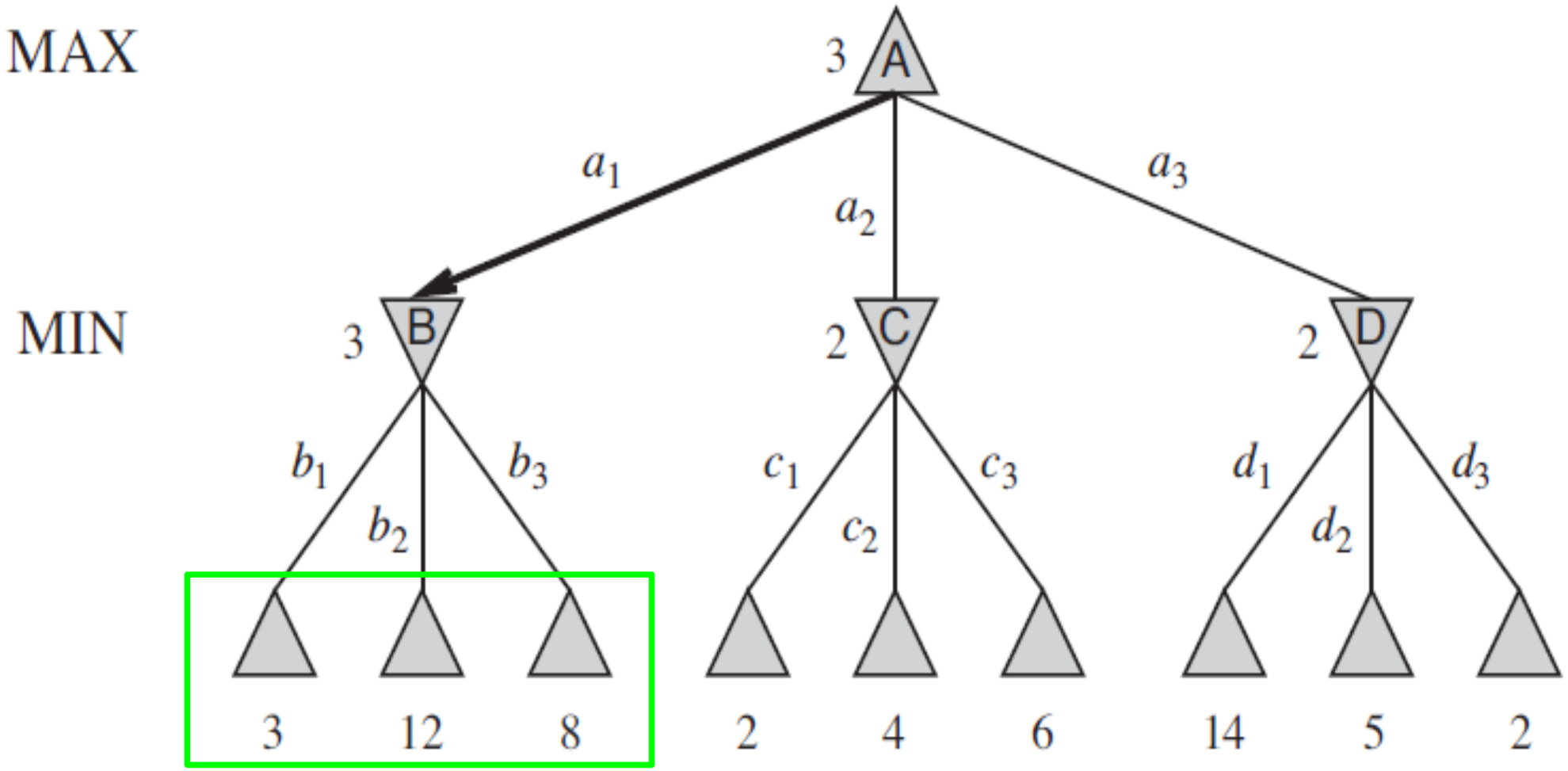
  $= \max(\text{Minimax}(\text{Result}(s,a)))$ if $\text{Player}(s) = \text{Max}$

  $= \min(\text{Minimax}(\text{Result}(s,a)))$ if $\text{Player}(s) = \text{Min}$

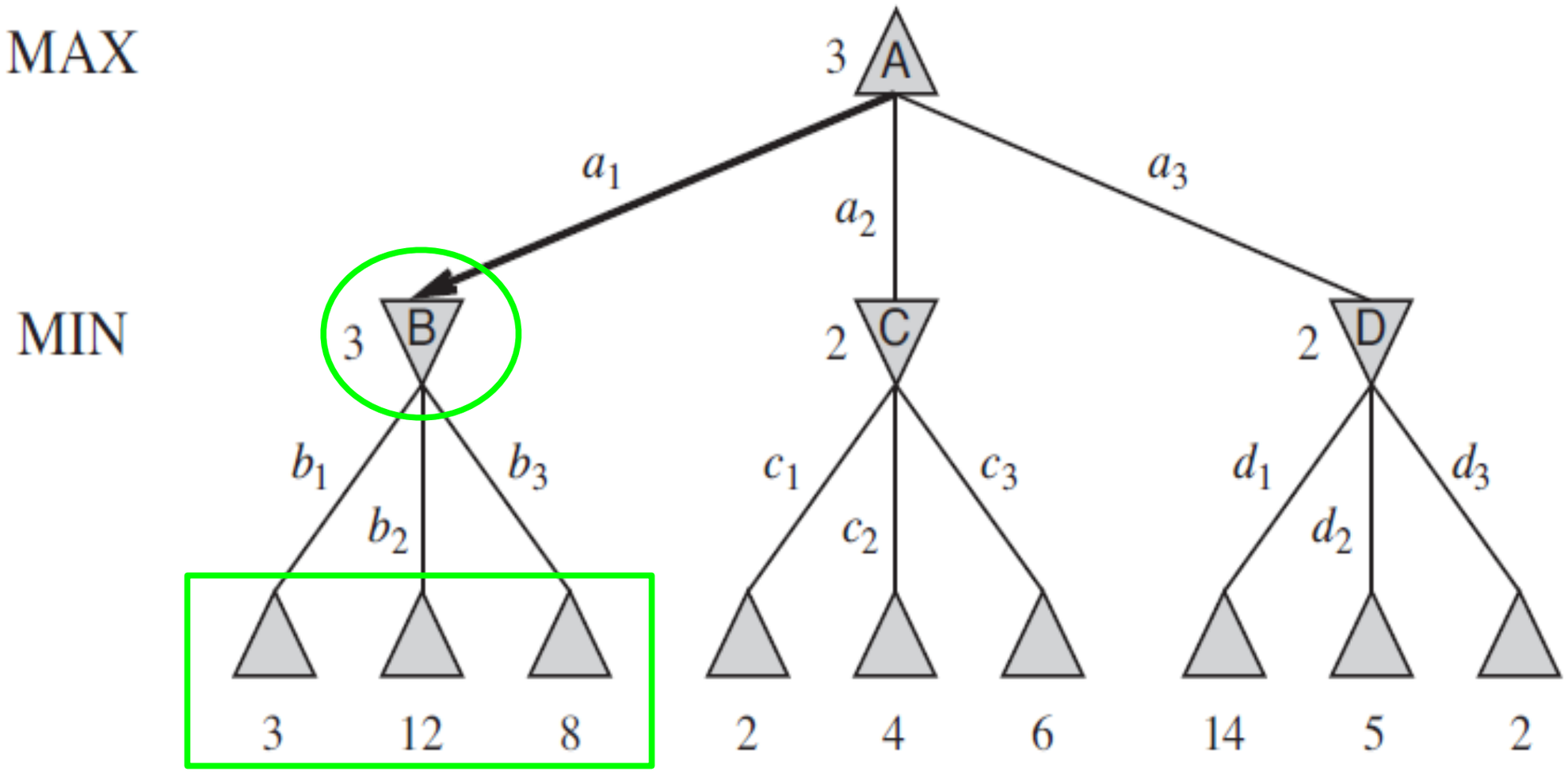
MINIMAX: EXAMPLE



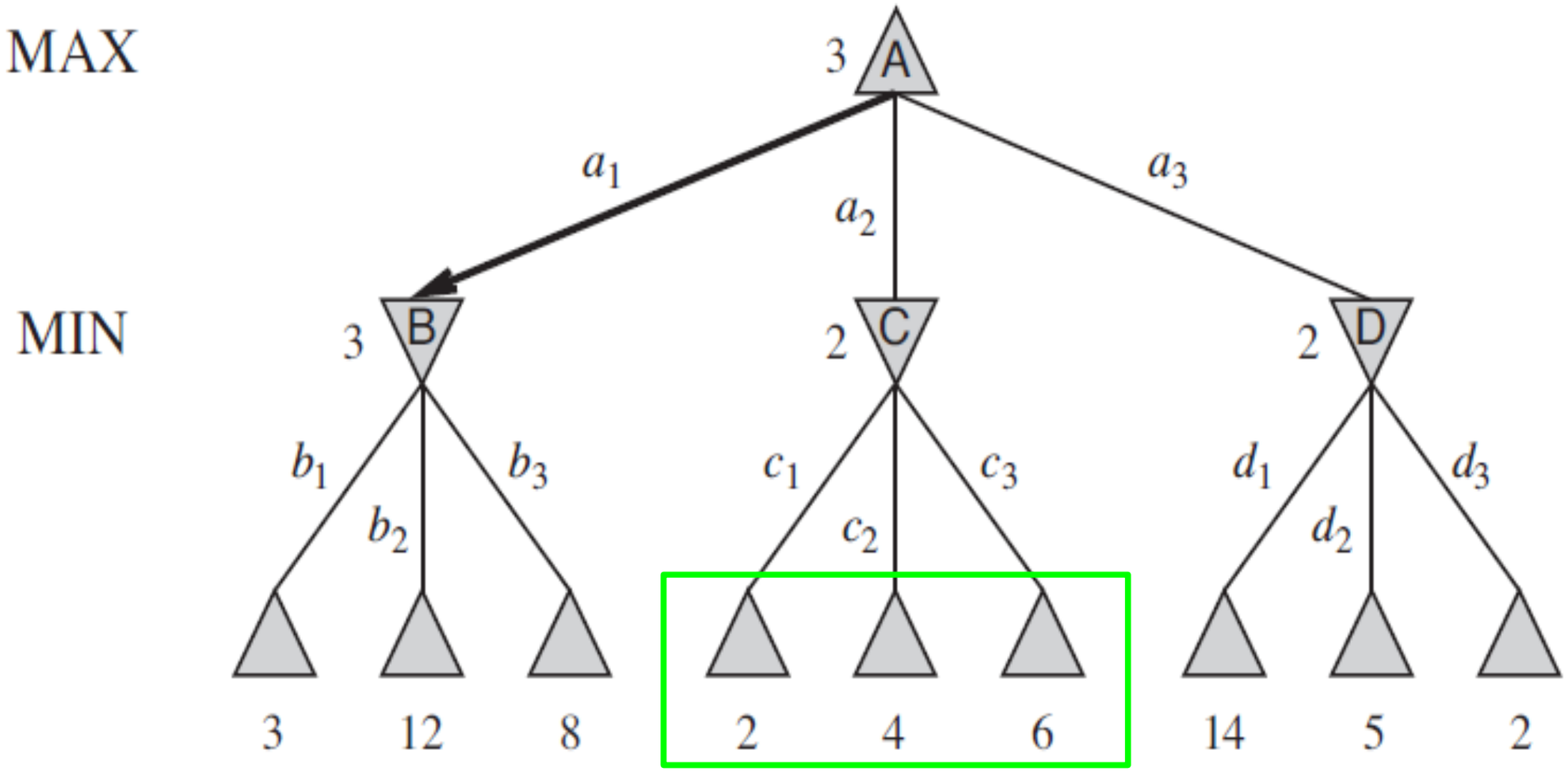
MINIMAX: EXAMPLE



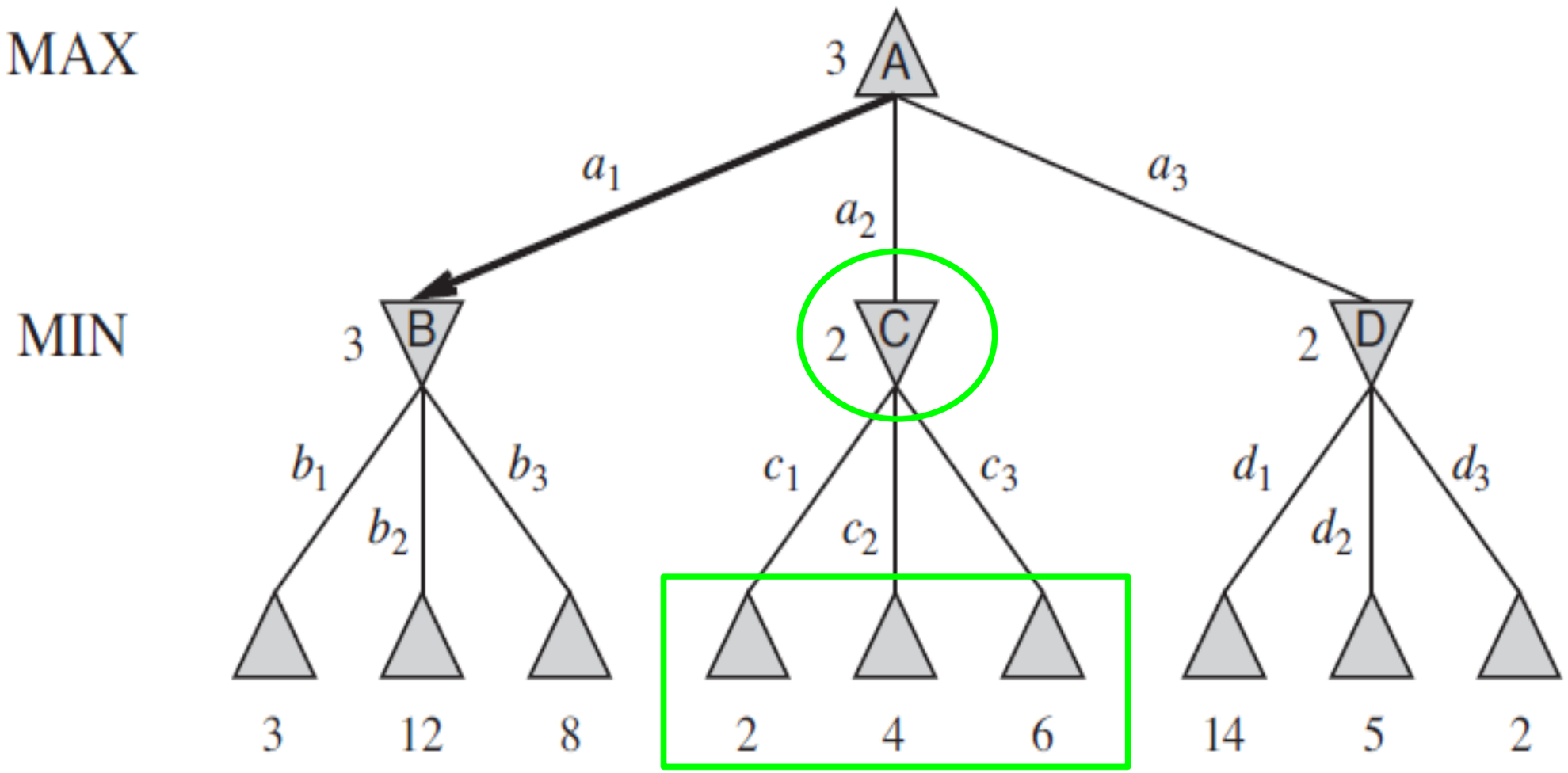
MINIMAX: EXAMPLE



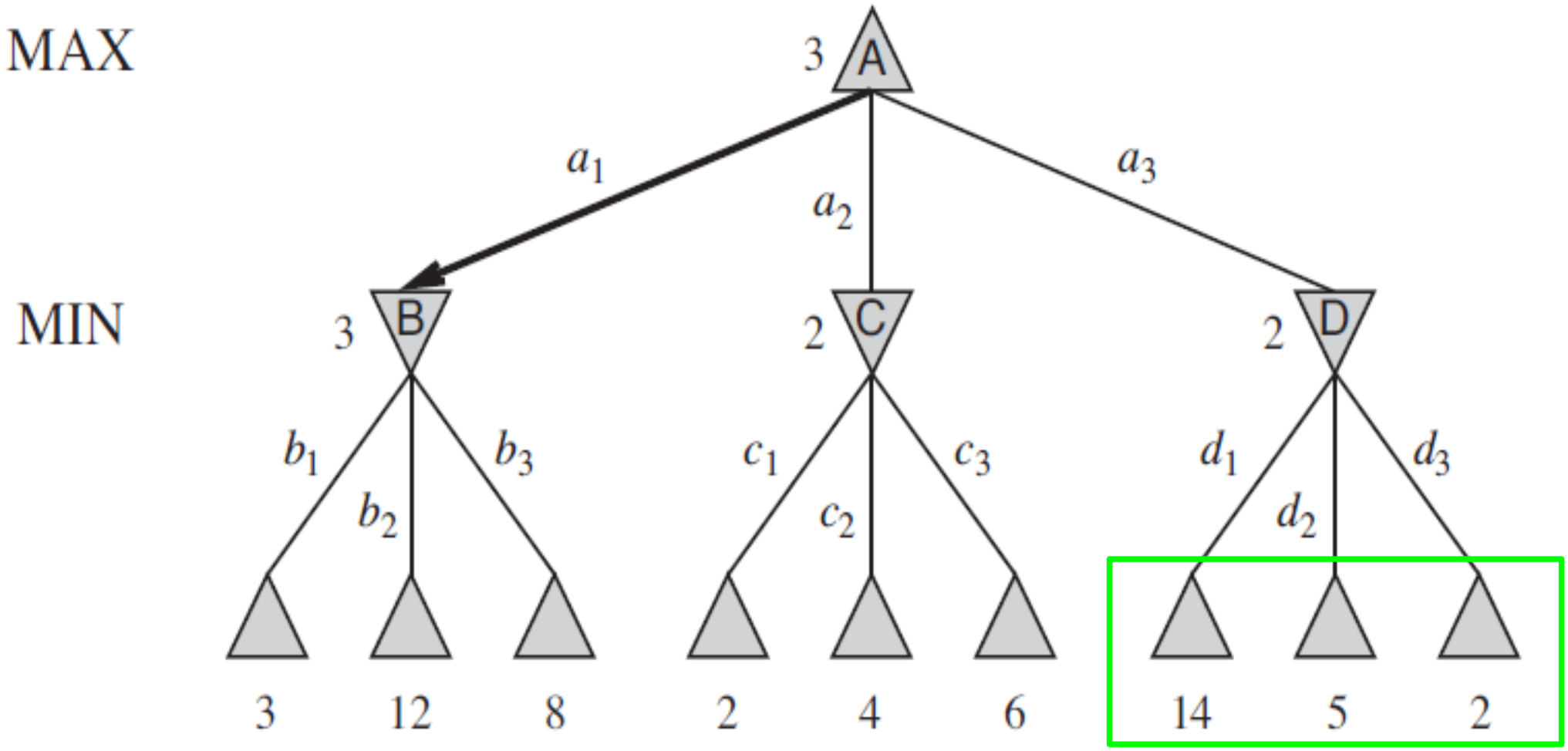
MINIMAX: EXAMPLE



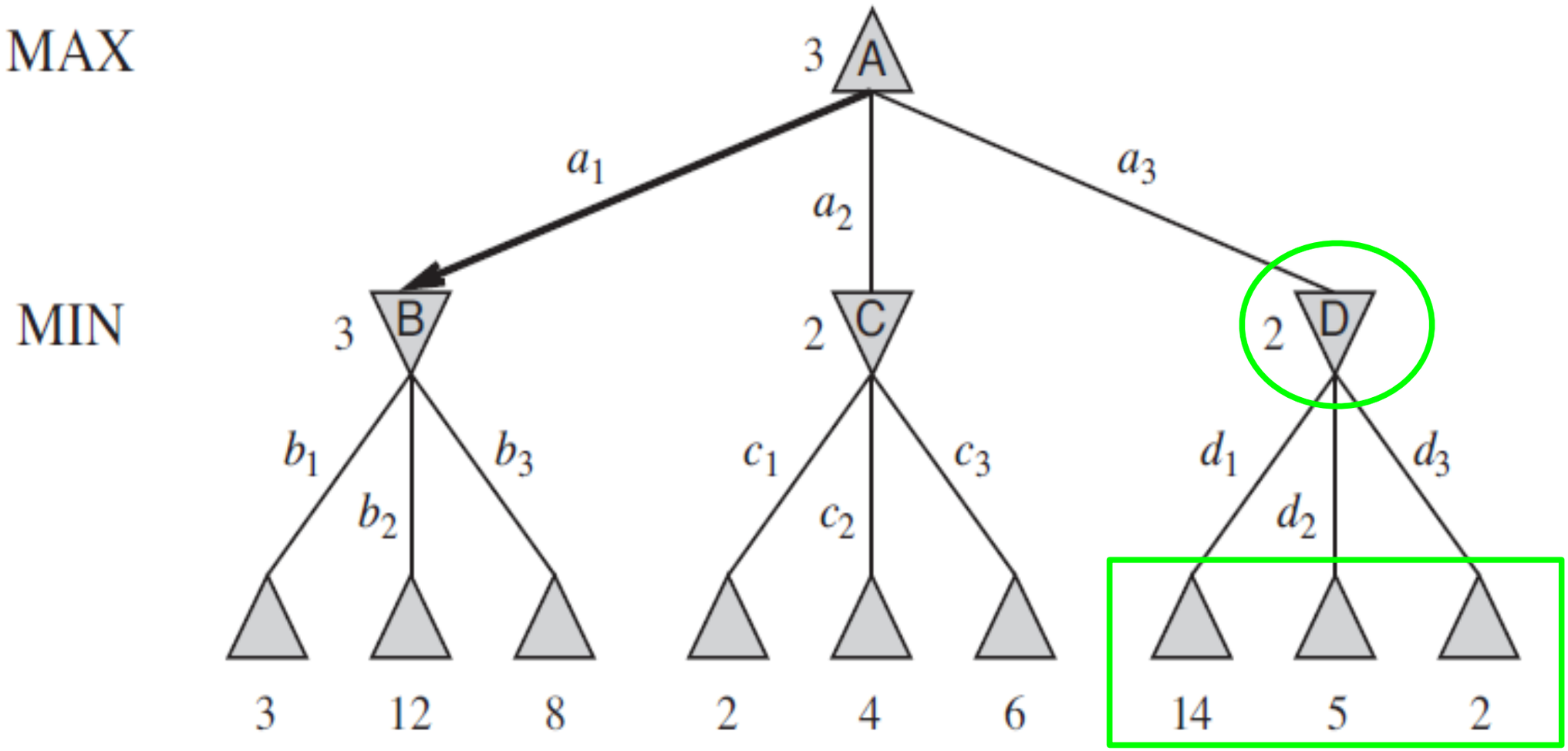
MINIMAX: EXAMPLE



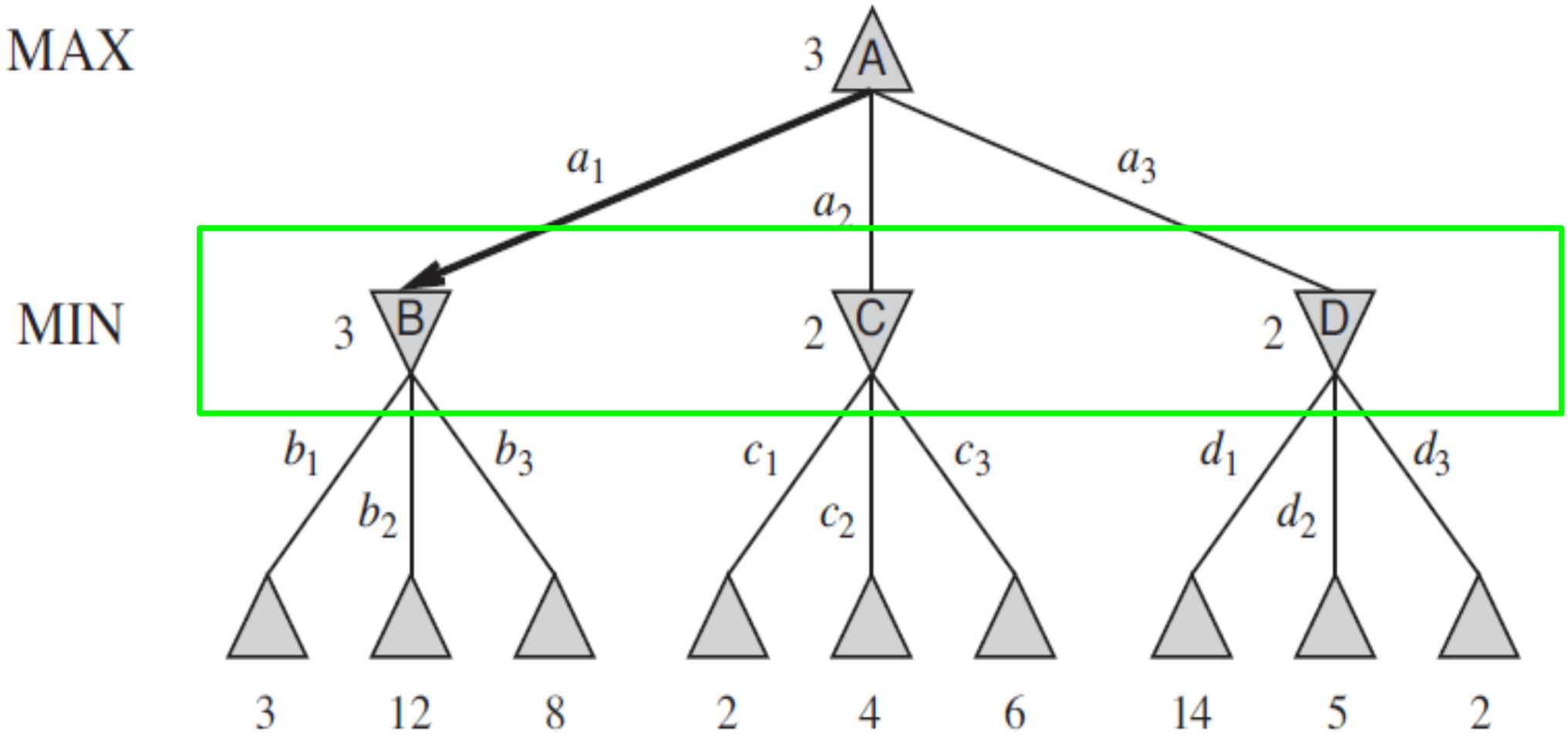
MINIMAX: EXAMPLE



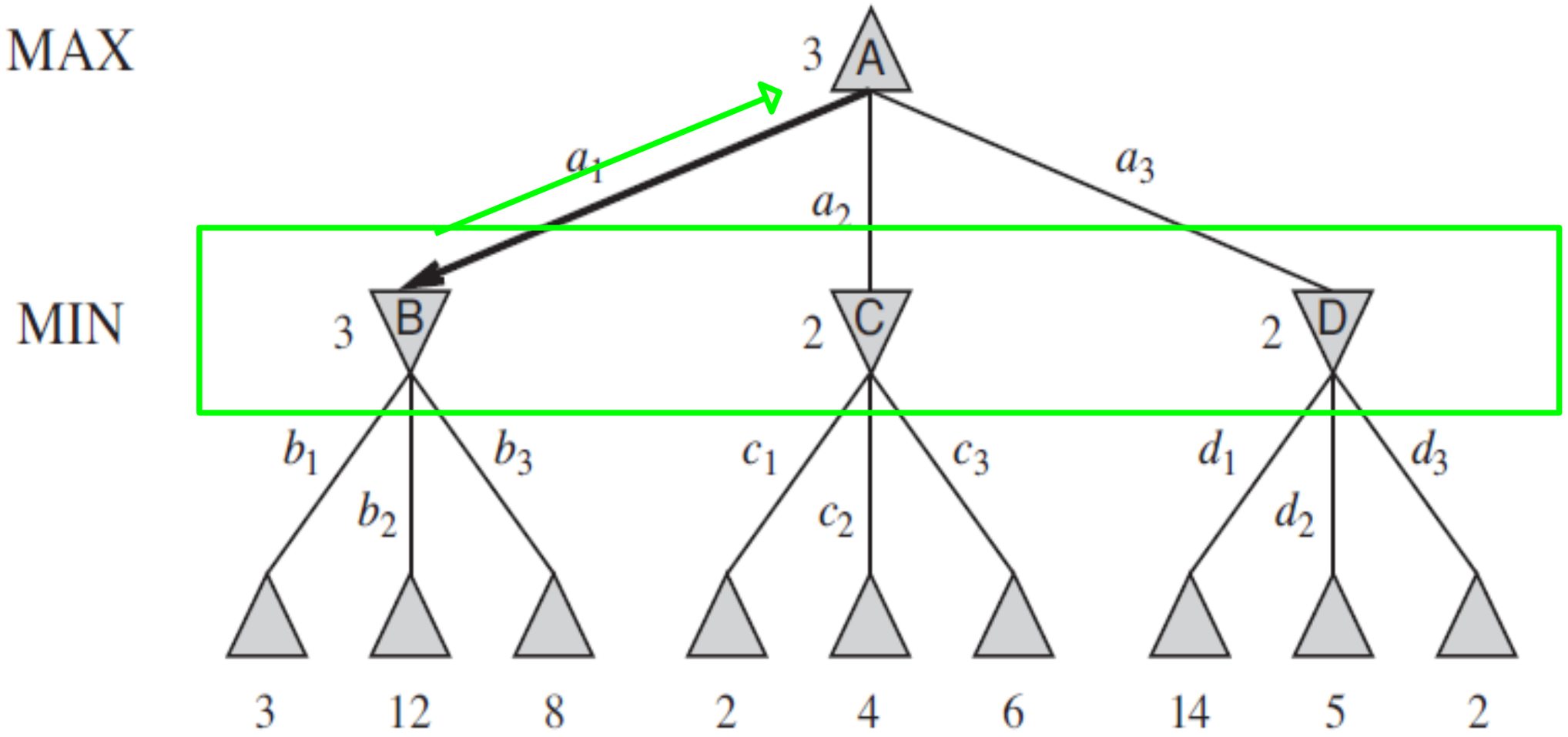
MINIMAX: EXAMPLE



MINIMAX: EXAMPLE



MINIMAX: EXAMPLE



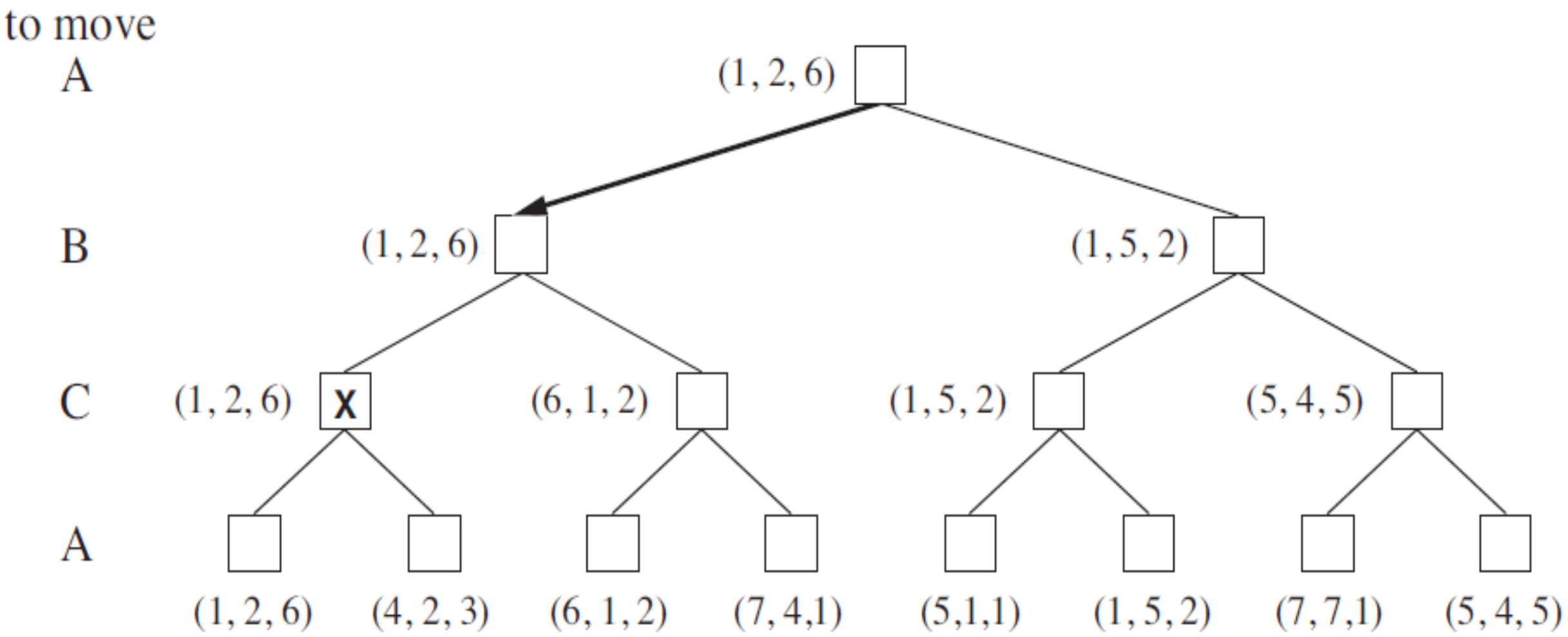
MINIMAX: THE ALGORITHM

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

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

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

MULTIPLAYER GAME



ONLINE SEARCH

Experience the Unknown

OFFLINE VS ONLINE

In offline search, we can compute a complete solution before entering the real world.

Online search computes actions on the go

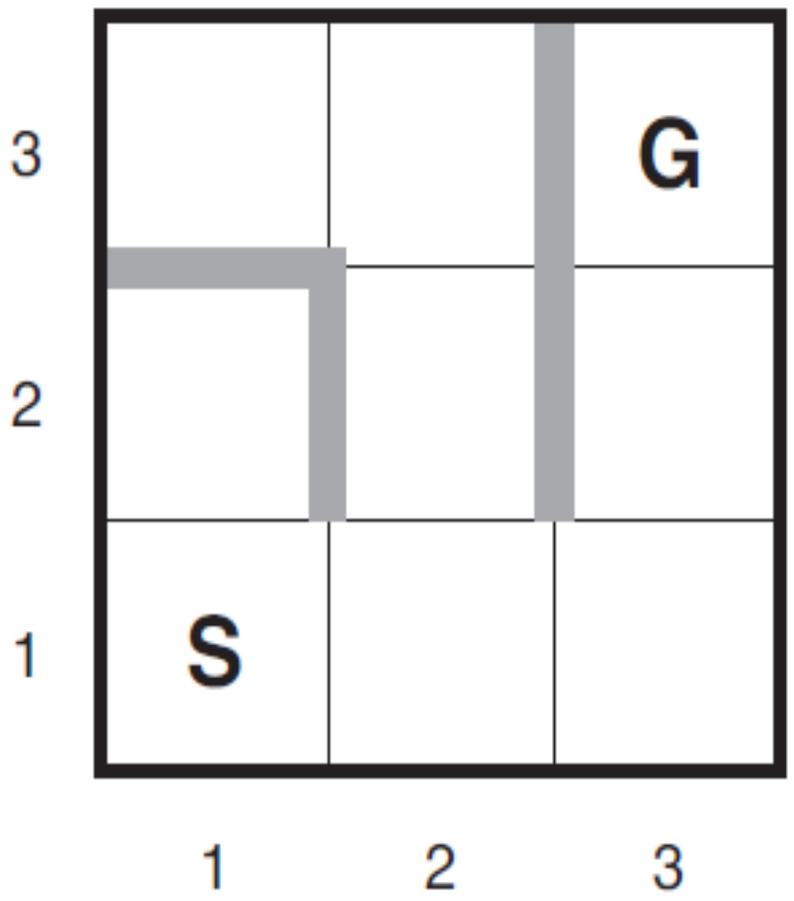
- Take action
- Observe environment
- Goal test
- Plan next action

ONLINE SEARCH EXAMPLE

S = Starting Position

G = Goal State

We need to find an algorithm to "sear
to G.



ONLINE SEARCH: DFS ALGORITHM

function ONLINE-DFS-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table indexed by state and action, initially empty
    untried, a table that lists, for each state, the actions not yet tried
    unbacktracked, a table that lists, for each state, the backtracks not yet tried
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in *untried*) **then** *untried*[s'] $\leftarrow$ ACTIONS(s')
  if s is not null **then**
    result[s, a] $\leftarrow$ s'
    add s to the front of *unbacktracked*[s']
  if *untried*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then return** *stop*
    else $a \leftarrow$ an action b such that *result*[s', b] = POP(*unbacktracked*[s'])
  else $a \leftarrow$ POP(*untried*[s'])
  $s \leftarrow s'$
  return a

ONLINE SEARCH EXAMPLE

function ONLINE-DFS-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: *result*, a table indexed by state and action, initially empty
    untried, a table that lists, for each state, the actions not yet tried
    unbacktracked, a table that lists, for each state, the backtracks not yet tried
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state (not in *untried*) **then** *untried*[s'] ← ACTIONS(s')
  if s is not null **then**
    result[s, a] ← s'
    add s to the front of *unbacktracked*[s']
  if *untried*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then return** *stop*
    else a ← an action b such that *result*[s', b] = POP(*unbacktracked*[s'])
  else a ← POP(*untried*[s'])
  s ← s'
  return a