# Informed Search - Exploration & Strategies

# Contents

- **Introduction**
  - Heuristic Function
- **Informed Search Strategies**
  - Greedy Best First Search
  - A* Search

# Introduction

Informed Search Strategy uses Problem-Specific Knowledge or Domain Knowledge along with the Problem definition to find Solutions More Efficiently than Uninformed search.

**Heuristic Function:**

A node is selected for expansion based on the Evaluation Function *F(n)*. The node with "lowest evaluation" is expanded first.

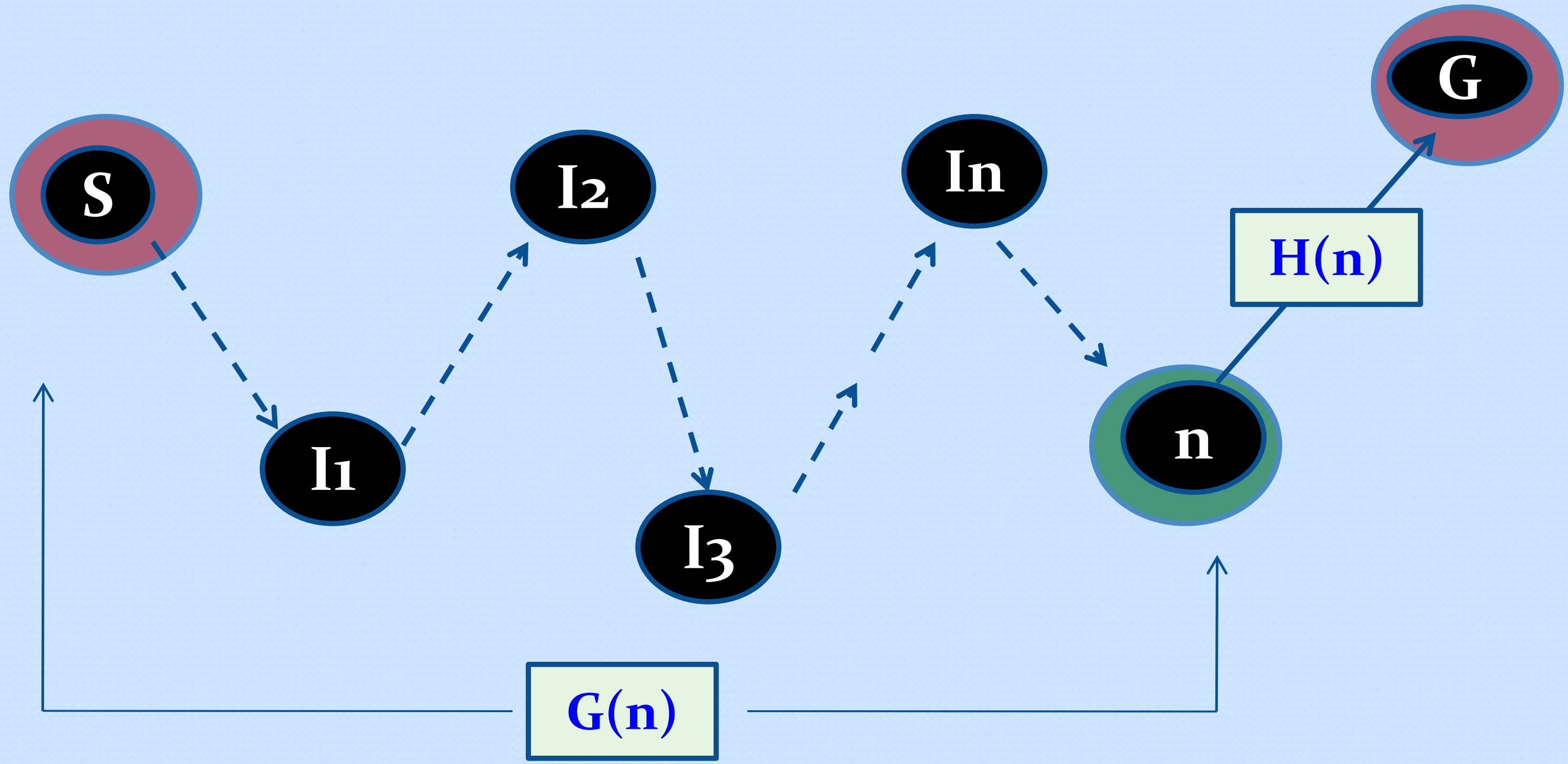
F(n) = H(n) + G(n)

H(n) is called Heuristic function that is the estimation of cheapest path to goal node 'n'. At goal node H(n) = 0.

*G(n)* is the no. of nodes travelled from start node to current node.

# Heuristic Function

- ***F(n) = H(n) + G(n)***
- ***H(n)*** – How far the **goal** G is
- ***G(n)*** – No. of nodes travelled from **start** node S to **current** node n



# Domain Knowledge & Heuristic Function

- **Domain Knowledge** is used for :
  - For **guiding** the **search** & Generating next states
  - **Heuristics** uses "domain specific knowledge" to estimate the "quality of potential solutions"

- **Examples of Heuristic Functions:**
  - **Manhattan distance** heuristic for "**8 puzzle**"
  - **Minimum Spanning Tree** heuristic for "**TSP**"
  - Heuristics are fundamental to "**Chess programs**"

- A strategy is defined by **picking** *the order of node expansion*

# 8-Puzzle (Initial & Goal States)

## <u>Manhattan distance</u> heuristic

Initial State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| | 4 | 6 |
| 7 | 5 | 8 |

Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

<u>Operations</u>

1. Up
2. Down
3. Right or
4. Left

Step cost - 1

- ***F(n) = H(n) + G(n)***
- ***H(n)*** – How far the goal is – **NO. OF MISPLACED TILES (4)**
- ***G(n)*** – Number of nodes travelled from start node to current node

# The informed search problem

- **Problem definition:**
  - Given: **[S, s, O, G, h]** where
    - **S** is the set of states
    - **s** is the start state
    - **O** is the set of state transition operators having some cost
    - **G** is the set of goal states
    - **h( )** is a heuristic function estimating the distance to a goal
  - **Goal is to find:**
    - A minimum cost "sequence of transitions" to a goal state

# Evaluation Function (f(n))
## for different search algorithms

- ***f(n)* – *Evaluation function*** *(Estimated cost from **start node** to **goal node** thru **current node 'n'**)*
- ***h(n)*** – ***Heuristics*** – *Estimated cost of how far the **goal node** is from **current node 'n'***
- ***g(n)*** – *Cost so far from **start node** to **node 'n'***

- ***Greedy Search***
- ***f(n) = h(n)*** *// heuristic (estimated cost) to goal node*

- ***Uniform Cost Search***
- ***f(n) = g(n)*** *// path cost so far*

- ***A* Search***
- ***f(n) = h(n) + g(n)*** *// path cost so far + heuristic to goal node*

# ★Greedy – Best first search (pg-93)

- **Greedy search** is one of the "**Best First Search**" Algorithm

- **Greedy best-first search** algorithm always selects the path which appears **best** at that moment
  - *May not always find the optimal solution*

- A **greedy search algorithm** uses a **heuristic** for making locally optimal choices at each stage with the hope of finding a global optimum

- We will use of **Greedy search** to solve **the route-finding problem** from **Arad** to **Bucharest** (in Romania)

# Route Finding problem (Travelling Romania)

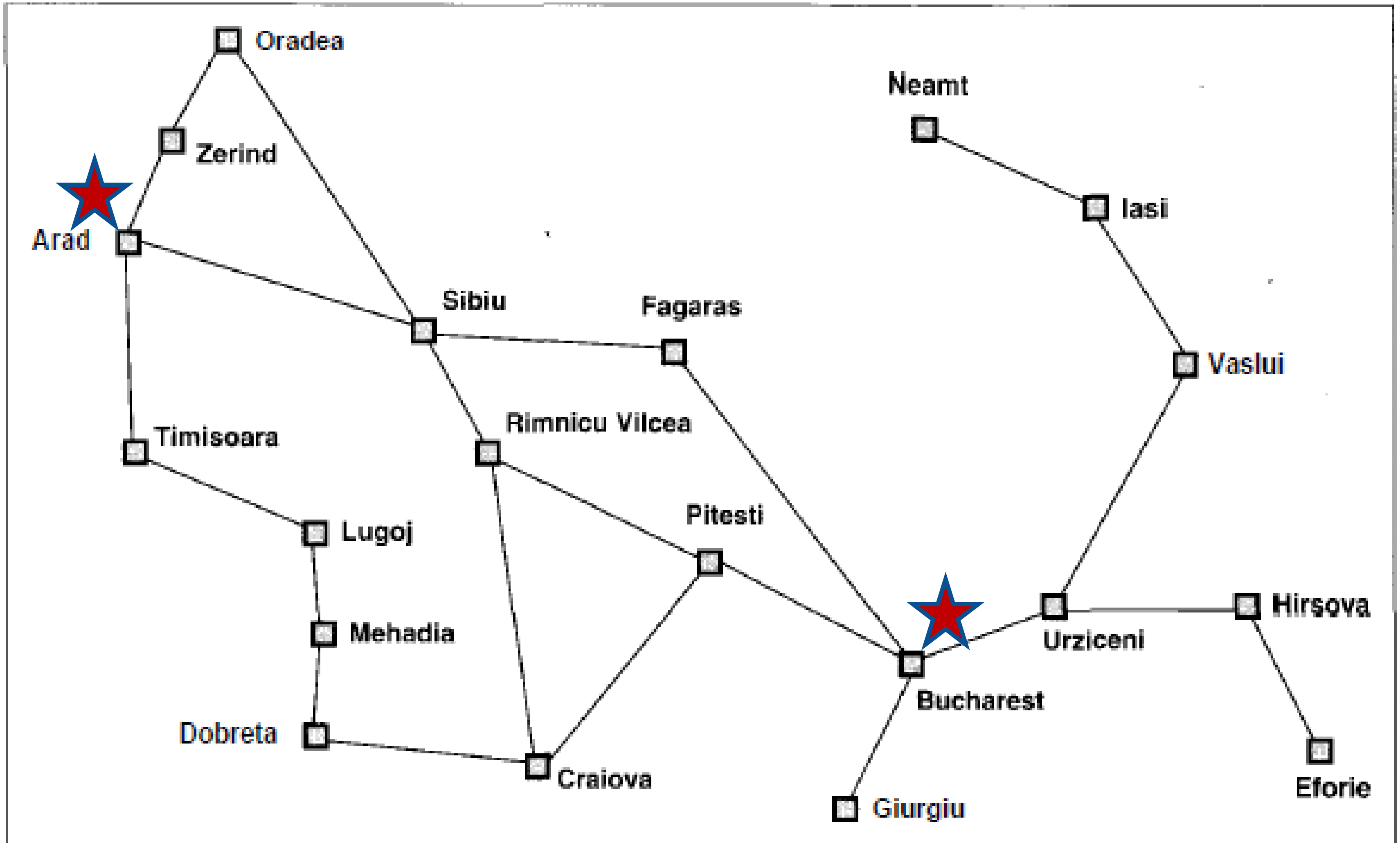



**Figure 3.3** A simplified road map of Romania.

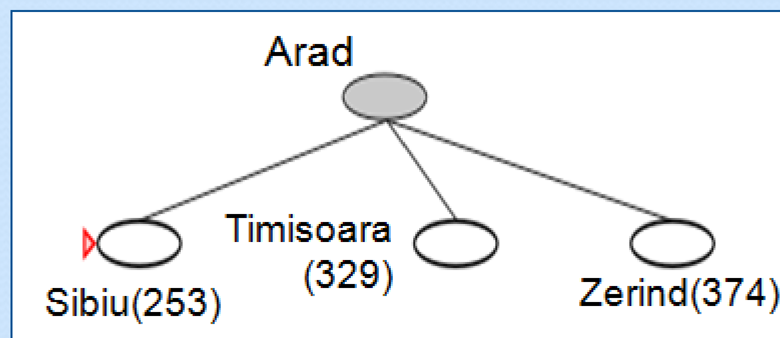
# ⭐ Greedy – Best first search (pg-93)

- Use of [Greedy search](#) to solve **the route-finding problem** from **Arad** to **Bucharest** (in Romania)
- **Heuristic** used = $h_{sld}$ = straight line distance (based on experience)
- The initial state=**Arad**


Straight-line distance to Bucharest

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

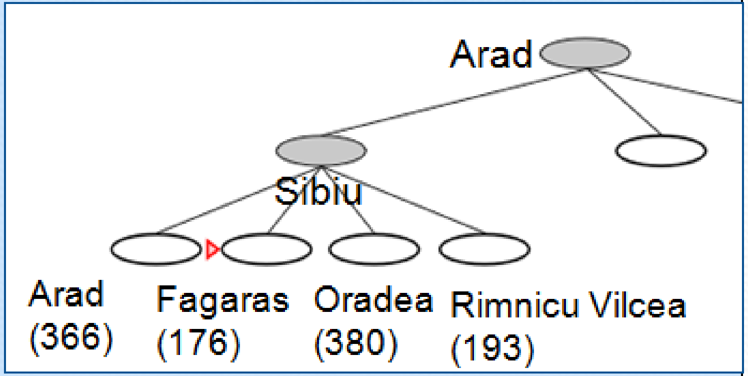
- $h_{sld}(In(Arad)) = 366$
- The first expansion step produces:
  - Sibiu, Timisoara & Zerind




- **Greedy best-first** will select **Sibiu**
  - As the heuristic ($h_{sld}$) is minimum

# **Greedy** – Best first search

- **Greedy best-first** will select **Sibiu**

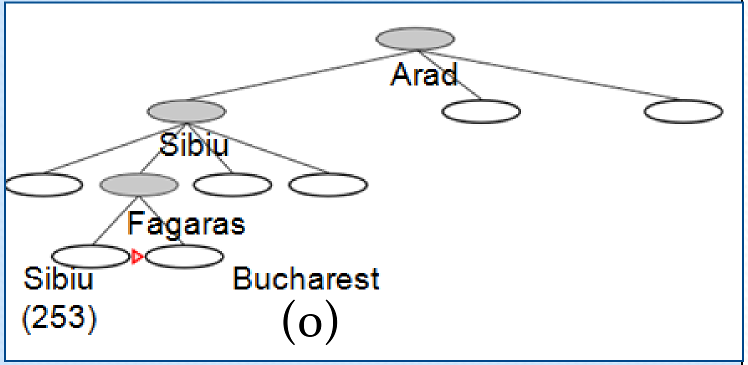



- If **Sibiu** is expanded we get:

Arad, Fagaras, Oradea & Rimnicu Vilcea

- **Greedy best-first search** will select: **Fagaras**

- If **Fagaras** is expanded we get:
  - Sibiu & **Bucharest**

# Route Finding problem (Travelling Romania)

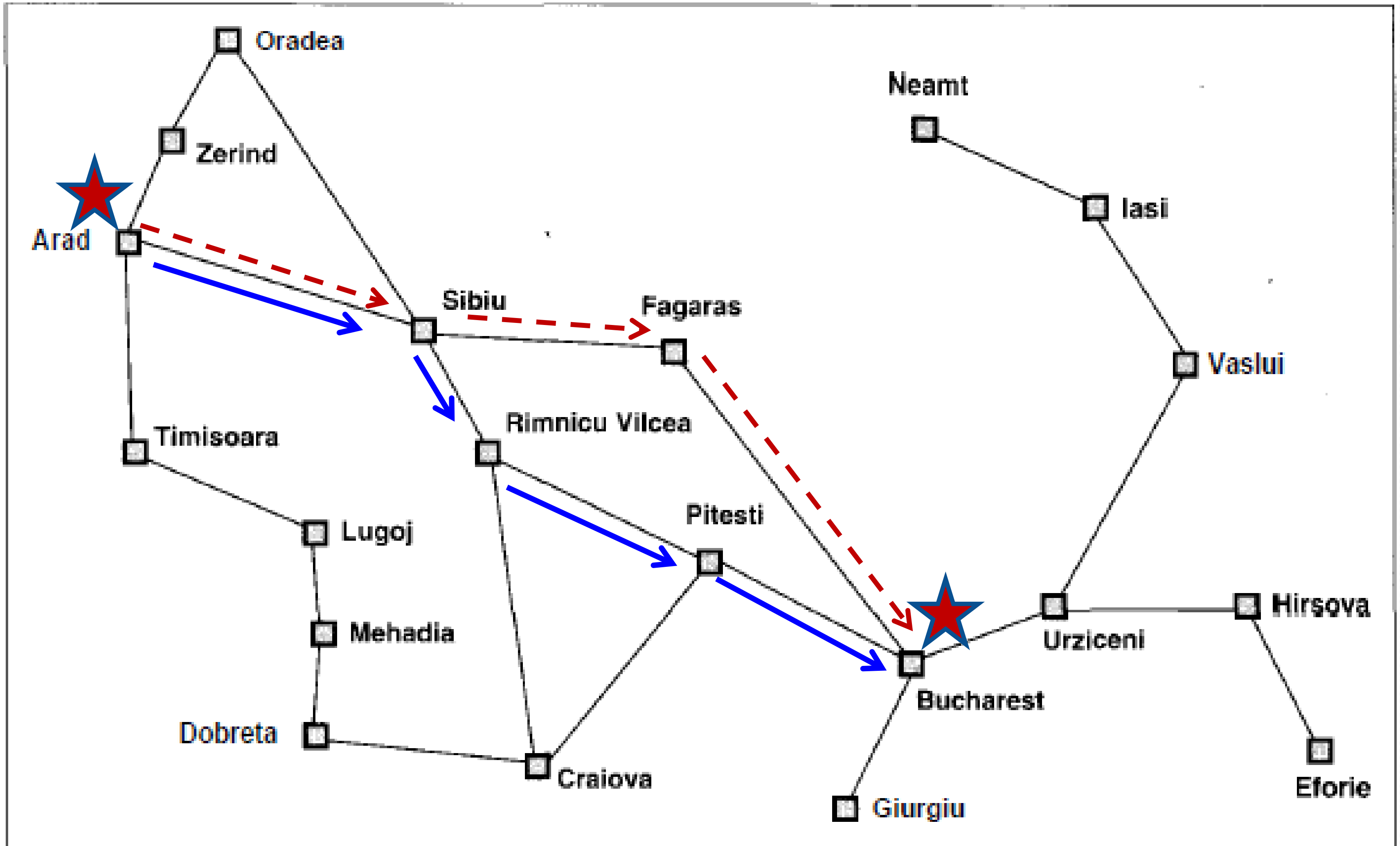



**Figure 3.3** A simplified road map of Romania.

# Greedy Best first search: Algorithm

1. Start with the initial state.
2. Until a goal is found or there is no nodes left
   2.1. Pick the best node
   2.2. Generate its successors
   2.3. for each successor:
   2.3.1. If it has not been generated before, evaluate it and record its parent.
   2.3.2. if it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

# **Greedy** – Best first search

- **Goal reached !!!**
  - Search **cost** is **minimal** (only expanded the nodes on solution path)
  - But **not optimal** (also leads to false paths)
  - ***Ex***: *Path Arad, Sibiu, Rimnicu Vilcea, Pitesti is 32 km shorter*

- Time & Space complexity for Tree is :$O(b^m)$
  - m – max depth of the search tree

- With quality heuristics, space & time complexity can be reduced substantially

# A* Search - Algorithm

- **A* Search** is Best-known form of **best-first search**
- **Idea**: Avoid expanding already expensive paths
- **Evaluation function** $f(n) = g(n) + h(n)$
  - $f(n)$ - Total cost of path from start node to "goal" through "$n$"
  - $g(n)$ - The cost of reaching the node "n" from "start"
  - $h(n)$ - Estimated cost of the cheapest solution through node "n" to the goal
- In order to get the **cheapest solution**, try the **node** with lowest value of $g(n) + h(n)$
- ***A* search** is – **Complete & Optimal***

# A* Search - Algorithm

**STEP-1 (Initialize)**: <Initialize the node lists - **OPEN** (currently known but not evaluated) & **CLOSED** (already evaluated)>

*// Initially, only the start node is known & no node is evaluated*

- Set **OPEN** = {s} & **CLOSED** = { }
- **g(s)** = 0 *// The cost of reaching from start node to start node is zero*
- **f(s)** = h(s) *// For first node, the evaluation function is completely based on heuristic*

**STEP-2 (Fail)**: <If nothing in **OPEN** node list, then terminate with failure>

- If **OPEN** = { }, **then terminate with failure**

**STEP-3 (Select)**: <If **OPEN** list has nodes, then select the cheapest cost node>

- Select the **minimum cost node n** from **OPEN** *//like done in route finding problem*
- Save **n** in **CLOSED** *// Move n into already evaluated list 'CLOSED'*

**STEP-4 (Goal test)**: <if node n is the **goal**, stop search, return Success & f(n)>

- If **n** $\in$ **G**, **terminate with success, return f(n)**

**STEP-5 (Expand)**: <Generate the successors of node n>

**For each successor m of n** // *For **each** of the **successors** do below steps*

// ***New Node*** *- If m is neither in OPEN or CLOSED => neither known nor evaluated before*

- If **m** $\notin$ [**OPEN** or **CLOSED**] **// g(m) is not yet calculated**

  // Calculate evaluation function f(m)

  - Set **g(m) = g(n) + Cost (n,m)** // *Find the distance from start to node m*
  - Set **f(m) = g(m) + h(m)** // *Evaluation function for node m is updated*
  - Insert **m** in **OPEN** // *Insert m in OPEN (known nodes list)*

// ***Old Node*** *- If m is either in OPEN or in CLOSED => either known or evaluated before*

- If **m** $\in$ [**OPEN** or **CLOSED**] **// g(m) is already calculated**
  - Set **g(m)** = **min {g(m), g(n) + Cost(n,m)}** // *set g(m) to lowest value till now*
  - Set **f(m) = g(m) + h(m)** // *Calculate evaluation function f(m)*

  // *If f(m) value of this node is less than earlier nodes at same level*

  - If **f(m)** has **decreased** & **m** $\in$ **CLOSED**
    - move it to **OPEN**

**STEP-6 Loop**: Go To Step 2.

# A* Search – Example-1

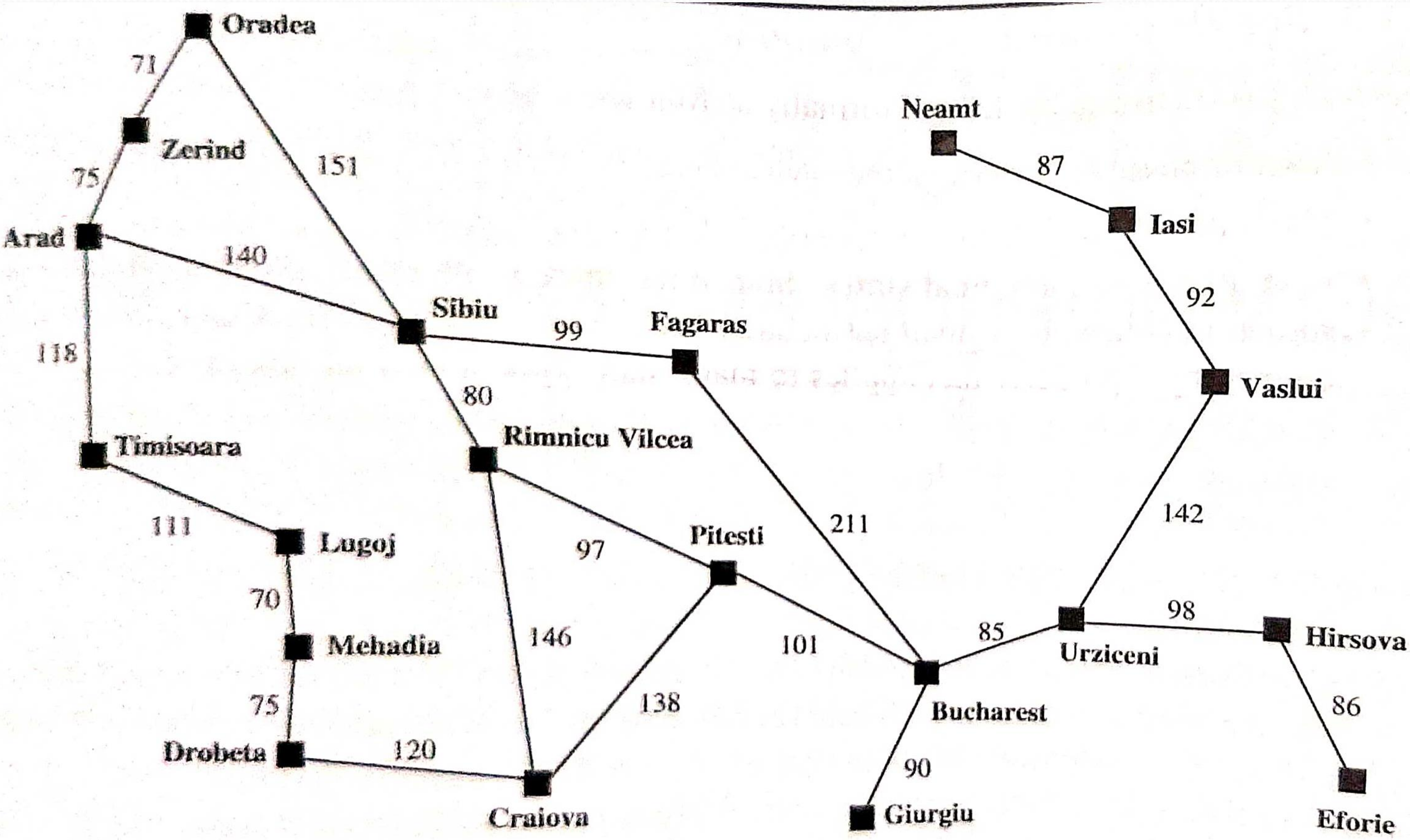


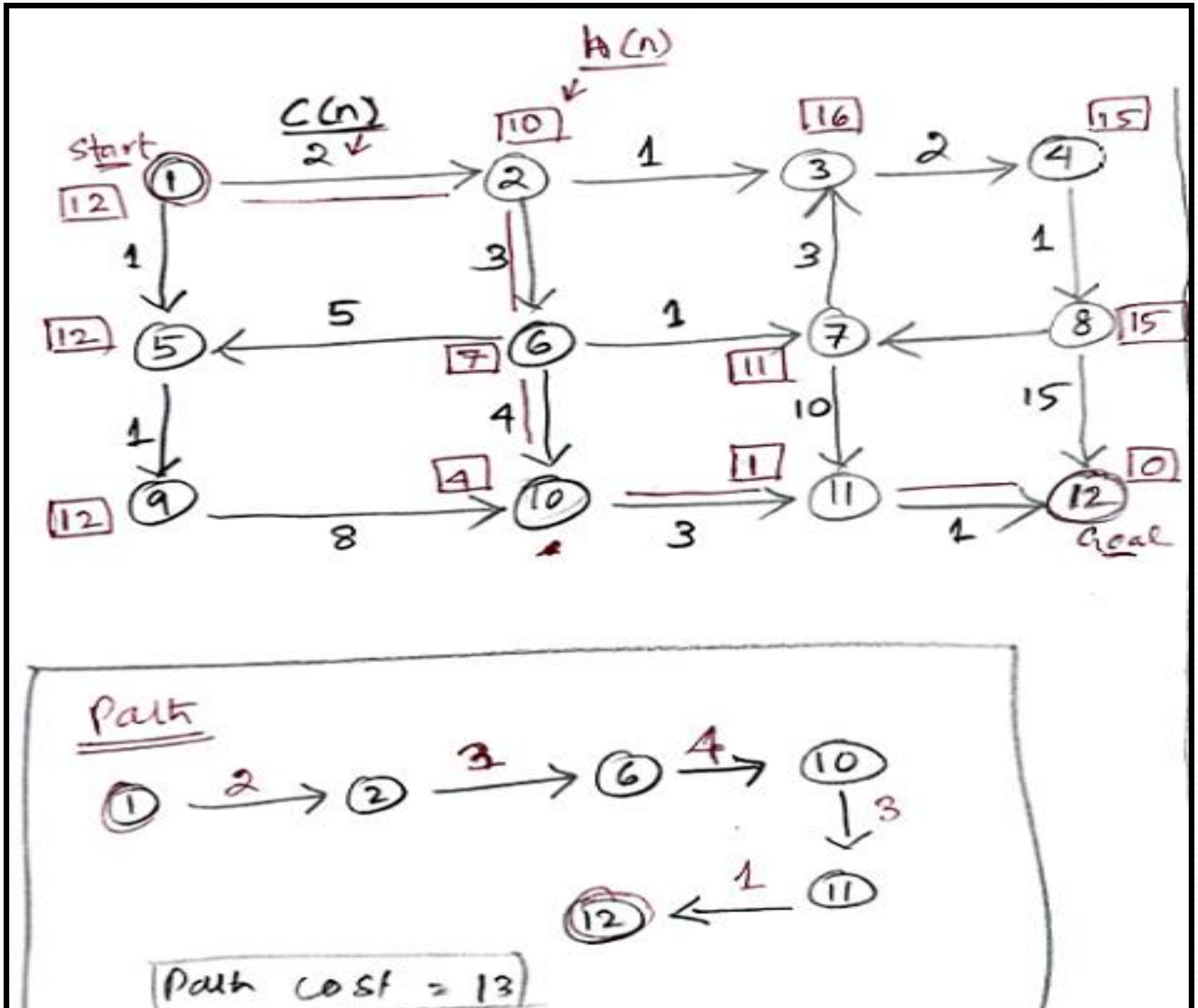

Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

# A* Search – Example-1

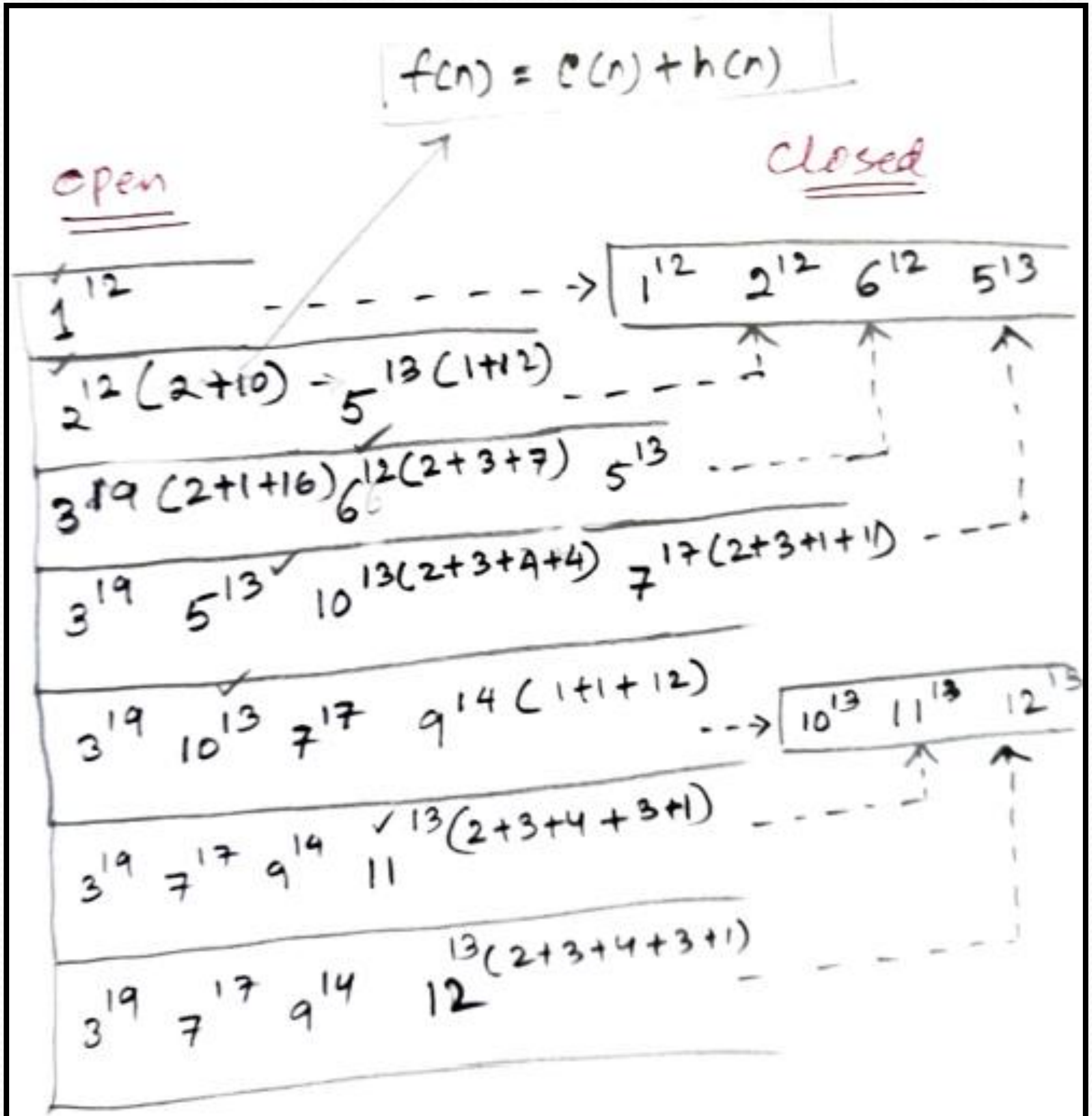
| | | | |
|---|---|---|---|
| **Arad** | 366 | **Mehadia** | 241 |
| **Bucharest** | 0 | **Neamt** | 234 |
| **Craiova** | 160 | **Oradea** | 380 |
| **Drobeta** | 242 | **Pitesti** | 100 |
| **Eforie** | 161 | **Rimnicu Vilcea** | 193 |
| **Fagaras** | 176 | **Sibiu** | 253 |
| **Giurgiu** | 77 | **Timisoara**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

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

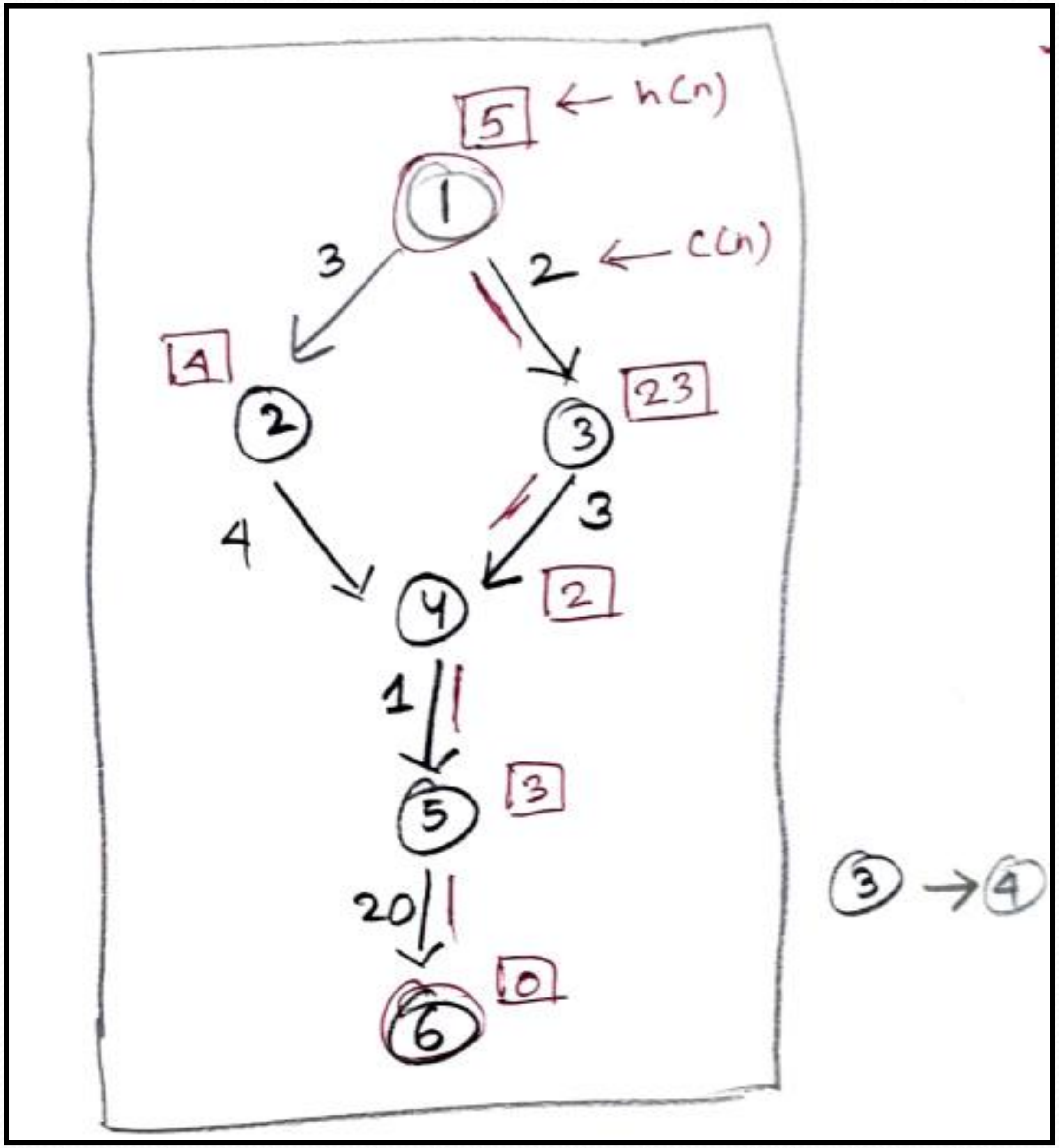
# A* Search – Example-2



# A* Search – Example-2



# A* Search – Example-3



# A* Search – Example-3

**Open**

$1^{5}$

$2^{7(3+4)}$ $3^{25(2+23)}$

$3^{25}$ $4^{9(3+4+2)}$

$3^{25}$ $5^{11(3+4+1+3)}$

$3^{25}$ $6^{28(3+4+1+20)}$

$4^{7(2+3+2)}$ $6^{28}$

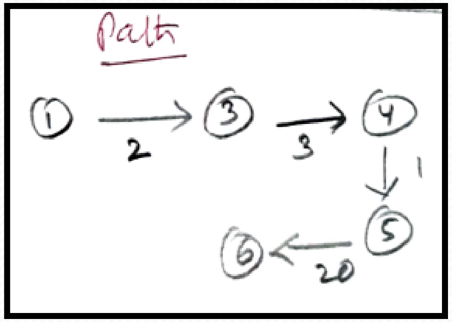
$6^{28}$ $5^{9(2+3+1+3)}$

$6^{26(2+3+1+20)}$

**Closed**

$1^{5}$ $2^{7}$ $\cancel{4^{9}}$ $\cancel{5^{11}}$ $3^{25}$ $4^{7}$ $5^{9}$

(we may have a better solution)



# 8-Puzzle (Initial & Goal States)

**Initial State**

| | | |
|---|---|---|
| 1 | 2 | 3 |
| | 4 | 6 |
| 7 | 5 | 8 |

**Goal State**

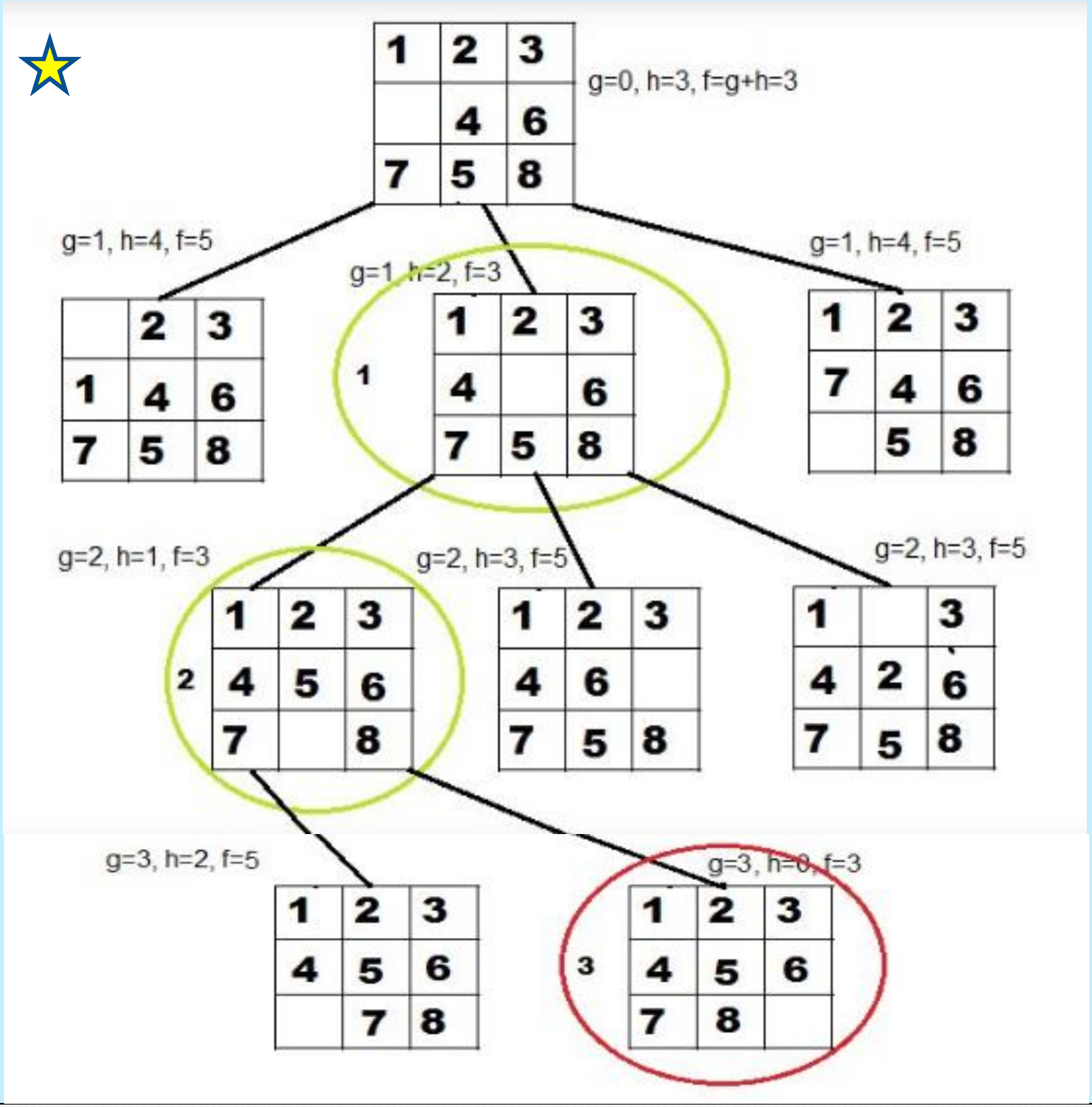
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Operations

1. Up
2. Down
3. Right or
4. Left

Step cost - 1

- F(n) = H(n) + G(n)
- H(n) – How far the goal is – No. of misplaced tiles
- G(n) – Number of nodes travelled from start node to current node




# 8-Puzzle (Solving steps)


The End