Solving Problems by Searching

CHAPTER 3

PREPARED BY

DR. ALIYA ALERYANI

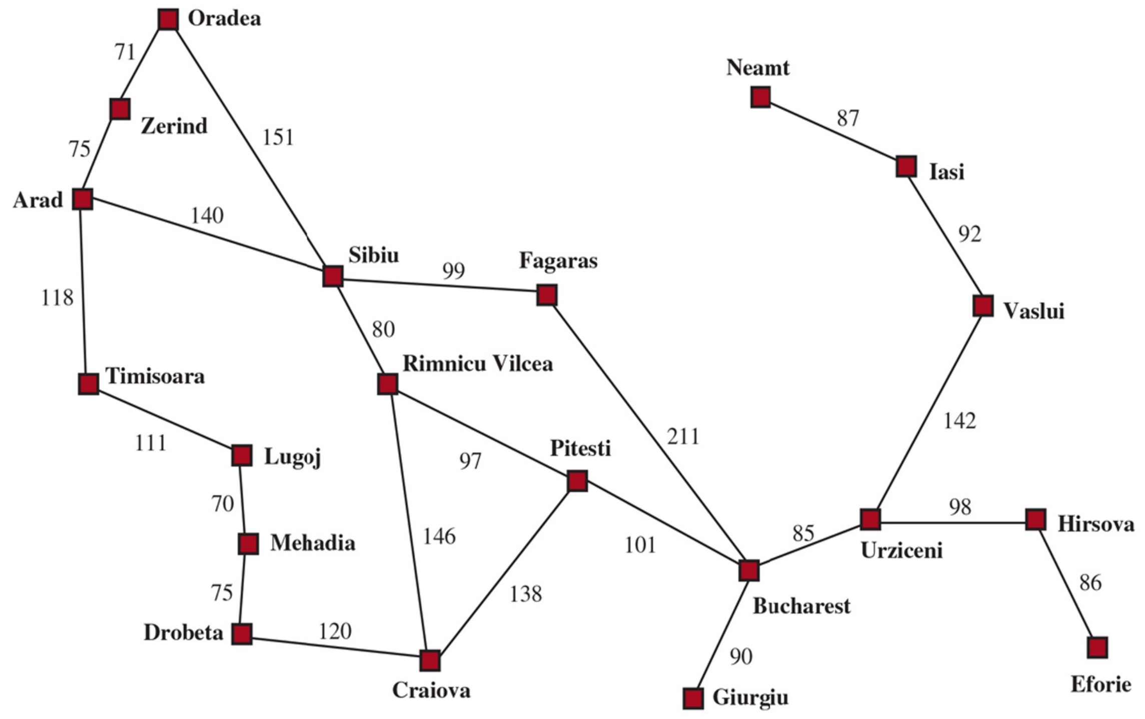
Solving Problems by Searching

- Problem-Solving Agents (Section 3.1)
- Example Problems (Section 3.2)
- Search Algorithms (Section 3.3)
- Uninformed Search Strategies (Section 3.4)
- Informed (Heuristic) Search Strategies (Section 3.5)

What is a Problem-Solving Agents?

- When the correct action to take is not obvious, an agent may need to to plan ahead.
- To consider a sequence of actions that form a path to a goal state, Such an agent is called a **problem-solving agent**.
- The computational process it undertakes is called **search**.

Agent in Romania



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

- If the agent has no additional information—the environment is unknown?
    - choose one action at random.
- Agent have access to information about the world
    - map

Problem-Solving Agents

The agent can follow this four-phase problem-solving process:

- **GOAL FORMULATION**: The agent adopts the goal of reaching Bucharest.
- **PROBLEM FORMULATION**: The agent devises a description of the states and actions necessary to reach the goal.
- **SEARCH**: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal.
- **EXECUTION**: The agent can now execute the actions in the solution, one at a time.

Search problems and solutions

A search problem can be defined formally as follows:

- A set of possible states that the environment can be in. We call this the **state space**.
- The **initial state** that the agent starts in.
- A set of one or more **goal states**.
- The **actions** available to the agent.
- A **transition model**, which describes what each action does.
- An **action cost** function.

Search problems and solutions

- A sequence of actions forms a **path**, and a **solution** is a path from the initial state to a goal state.
- The **total cost** of a path is the sum of the individual action costs.
- An **optimal solution** has the lowest path cost among all solutions.
- The state space can be represented as a **graph** in which the vertices are states and the directed edges between them are actions.

Formulating problems

- Our formulation of the problem of getting to Bucharest is a **model**—an abstract mathematical description.
- Only relevant states to the problem of finding a route to Bucharest are considered.
- The action “drive from Arad to Sibiu” can be carried out.

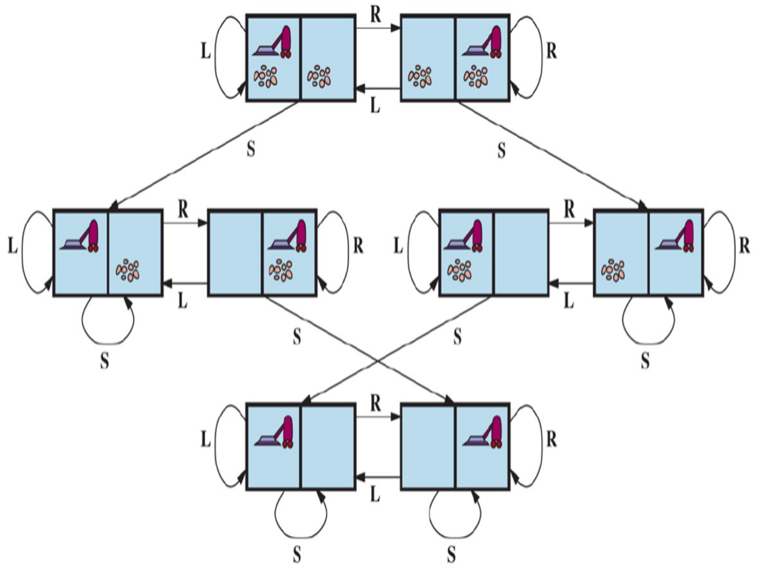
Example Problems

The vacuum world

It can be formulated as a grid world problem as follows:

- **STATES**: vacuum environment with n cells has $n.2^n$ sta
- **INITIAL STATE**: Any state can be as the initial state.
- **ACTIONS**: Suck, move Left, and move Right.
- **TRANSITION MODEL**: Suck removes any dirt;

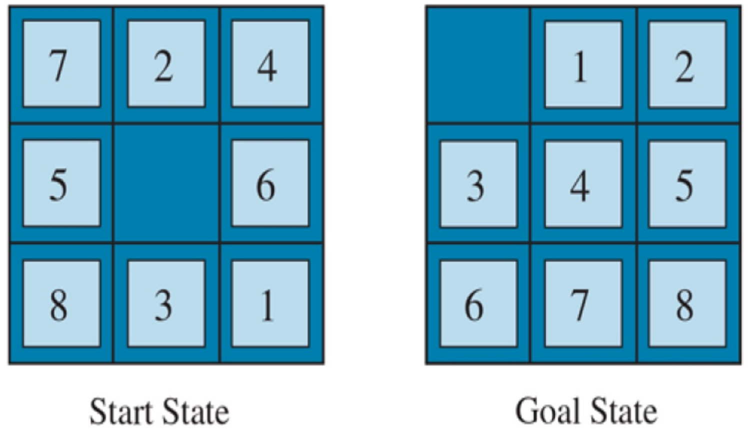
  Forward moves the agent ahead; Backward moves the agent in the opposite direction; TurnRight and TurnLeft change the direction it is facing by 90
- **GOAL STATES**: The states in which every cell is clean.
- **ACTION COST**: Each action costs 1.



The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = *Left*, R = *Right*, S = *Suck*.

Example Problems

- **Sliding-tile puzzle (8-puzzle)**
  - **STATES**: A state description specifies the location of each of the tiles.
  - **INITIAL STATE**: Any state can be designated as the initial state.
  - **ACTIONS**: the blank space moving Left, Right, Up, or Down
  - **TRANSITION MODEL**: Maps a state and action to a resulting state; for example, if we apply Left to the start state, the resulting state has the 5 and the blank switched.
  - **GOAL STATE**: a state with the numbers in order.
  - **ACTION COST**: Each action costs 1.



Real-world problems

- **Route-finding problem**

  Consider the airline travel problems that must be solved by a travel-planning Web site:

  - **STATES**: location (e.g., an airport) and the current time.
  - **INITIAL STATE**: The user’s home airport.
  - **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 new location and the flight’s arrival time as the new time.
  - **GOAL STATE**: A destination city.
  - **ACTION COST**: A combination of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, and so on.

Real-world problems

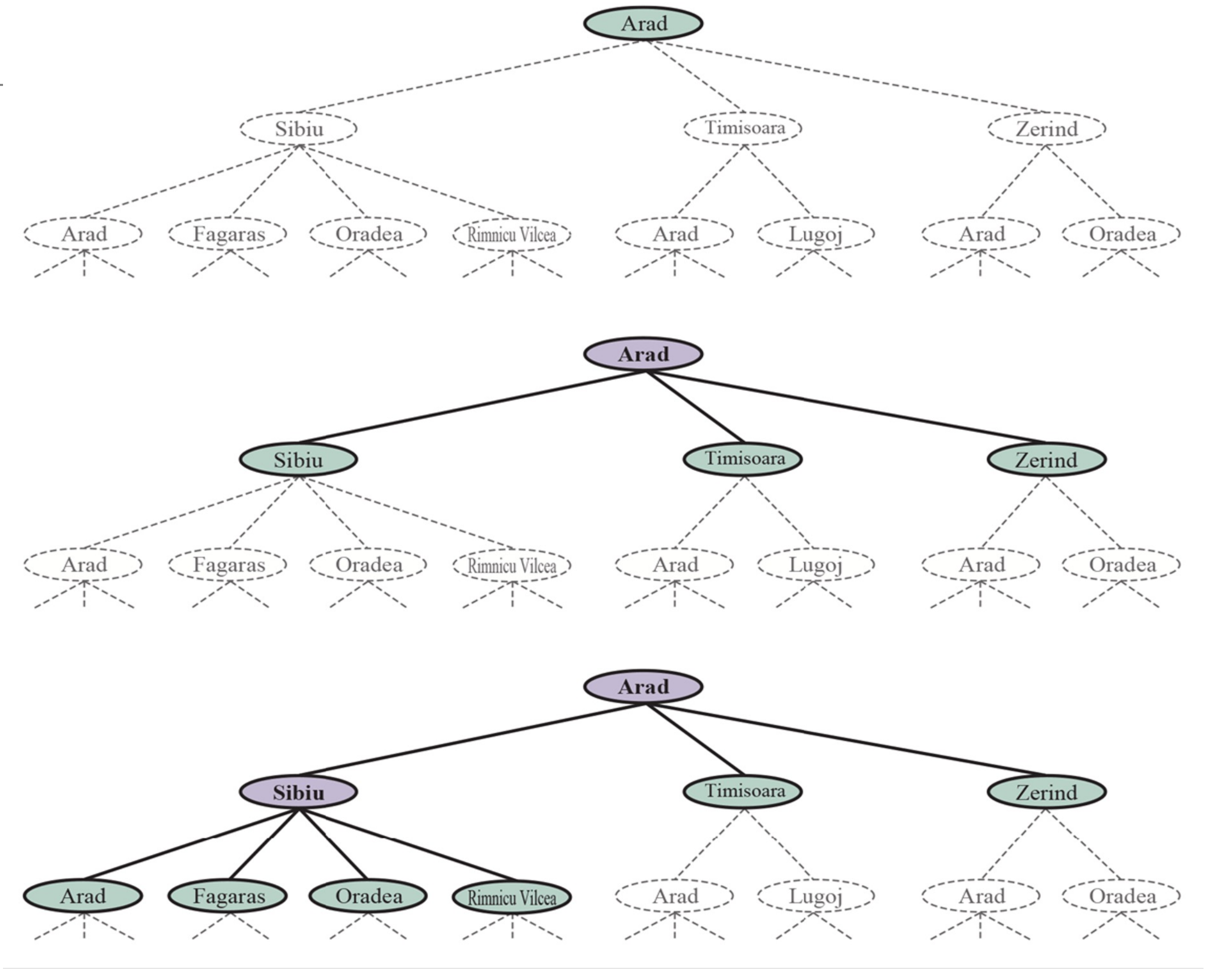
- **Touring problems** describe a set of locations that must be visited, rather than a single goal destination.
  - Exercise: Write the problem formulation for touring problem.
- **The traveling salesperson problem** (TSP): is a touring problem in which every city on a map must be visited.
  - The aim is to find a tour with cost < C (or in the optimization version, to find a tour with the lowest cost possible).
  - Exercise: Write the problem formulation for TSP.
- **Robot navigation:** a robot can roam around, in effect making its own paths.
  - Exercise: Write the problem formulation for Robot navigation.

Search Algorithms

- A search algorithm takes a search problem as input and returns a solution, or an indication of failure.
- Each **node** in the search tree corresponds to a state in the state space and the edges in the search tree correspond to actions.
- The **root** of the tree corresponds to the initial state of the problem.

Search Algorithms

- Nodes that have been expanded ?
- Nodes on the frontier that have been generated but not yet expanded?
- A cycle from Arad to Sibiu to Arad; can be an optimal path?



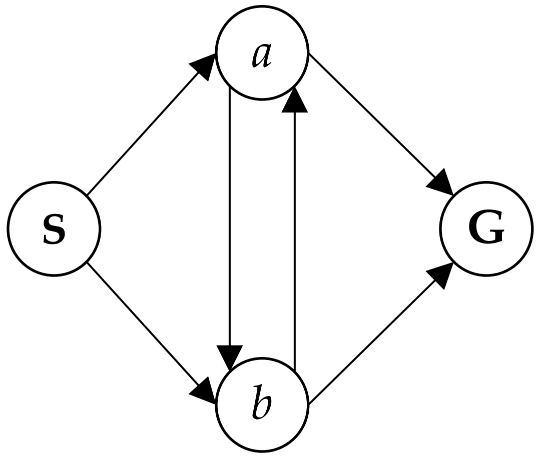
Measuring problem-solving performance

We can evaluate an algorithm’s performance in four ways:

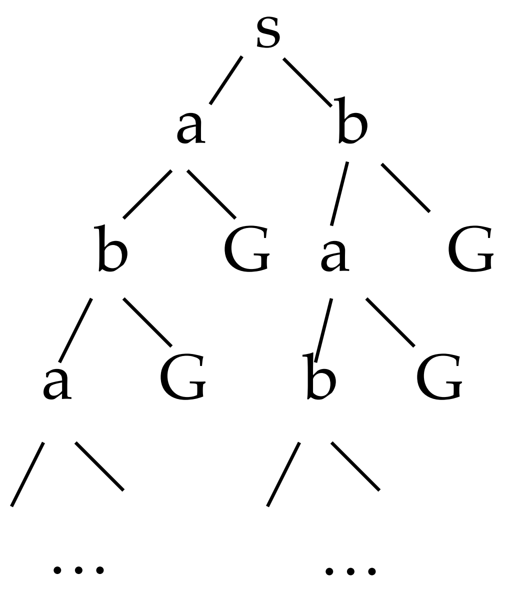
- **COMPLETENESS**: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- **COST OPTIMALITY**: Does it find a solution with the lowest path cost of all solutions?
- **TIME COMPLEXITY**: How long does it take to find a solution? This can be measured in seconds, or more abstractly by the number of states and actions considered.
- **SPACE COMPLEXITY**: How much memory is needed to perform the search?

State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



∞

Important: Lots of repeated structure in the search tree!

Uninformed Search Strategies

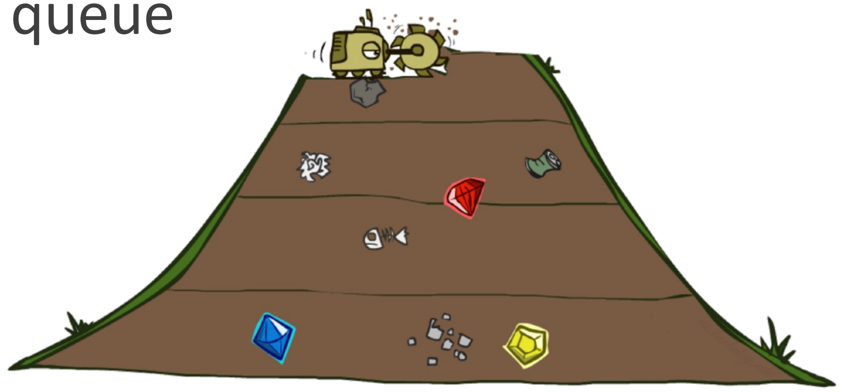
An uninformed search algorithm is given no clue about how close a state is to the goal(s) (blind searches) .

Examples:

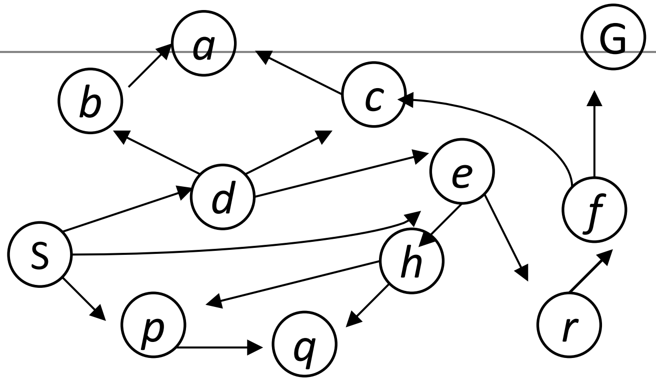
- Breadth-first search
- Depth-first search
- Dijkstra’s algorithm or uniform-cost search
- Depth-limited and iterative deepening search
- Bidirectional search

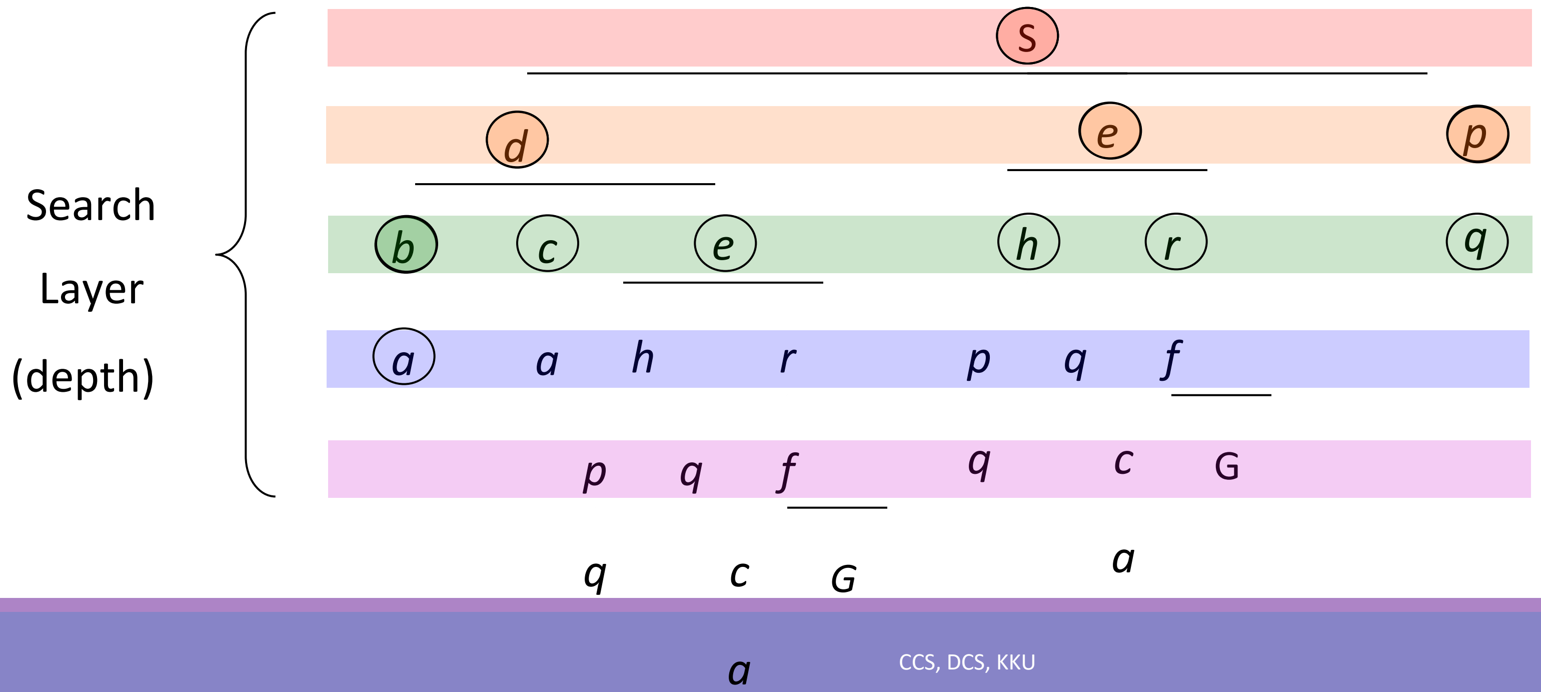
Breadth-First Search (BFS)

- An appropriate strategy when all actions have the same cost
- The root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on
- Places all new successors at the end of a standard FIFO queue



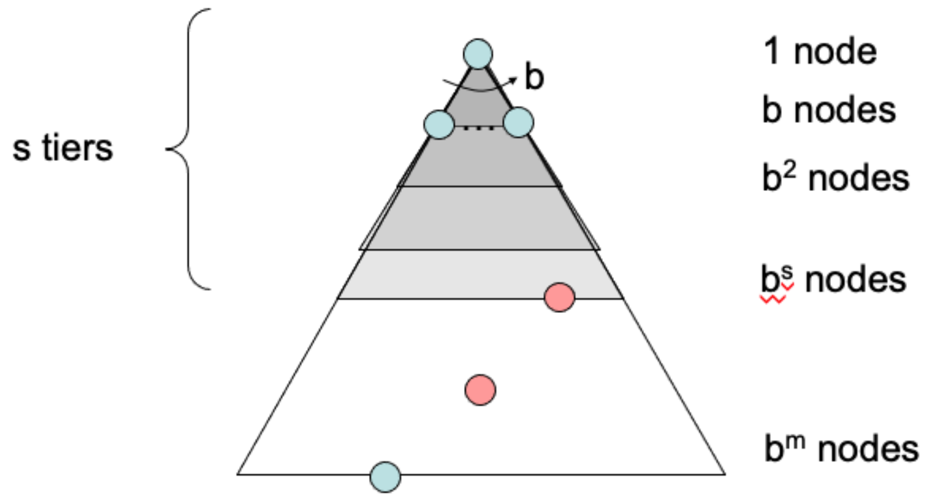
Breadth-First Search (BFS)





shallowest path?

Breadth-First Search (BFS)

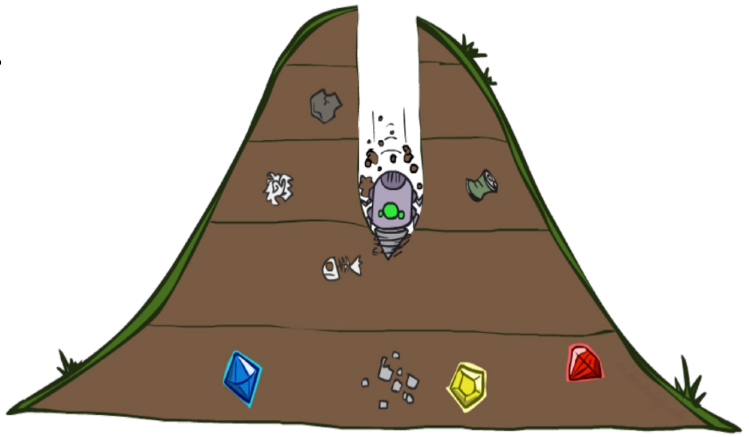


Breadth-First Search (BFS) Properties

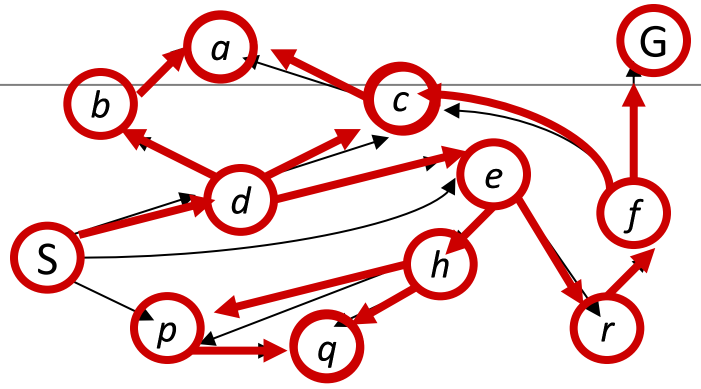
- Is it cost optimal?
  - It is cost-optimal only for problems where all actions have the same cost, but not for problems that don't have that property. BFS finds the shallowest path to any goal node. If multiple goal nodes exist, it finds the shortest path
- Is it complete?
  - It is complete always finds goal if one exists provided *b* (max **branching factor** which is number of successors of a node that need to be considered) is finite.
- Time complexity? Space complexity?
  - If root of the search tree generates *b* nodes, each of which generates b more nodes. Now suppose that the solution is at depth *s*, then both time and space complexity are $O(b^s)$

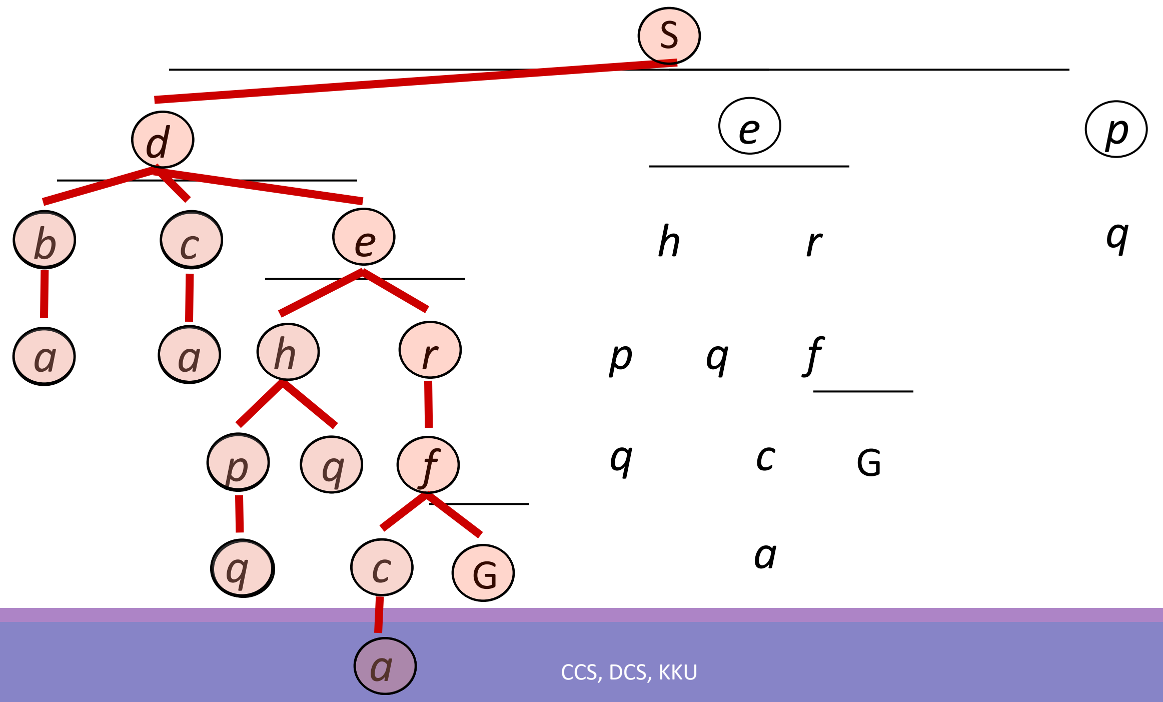
Depth-First Search (DFS)

- Always expands the deepest node in the frontier first.
- Implemented as a tree-like search.
- Search proceeds immediately to the deepest level of the search tree.
- The search then "backs up" to the next deepest node that still has unexpanded successors.
- Unexplored successors are placed on a stack (LIFO queue) until fully explored.



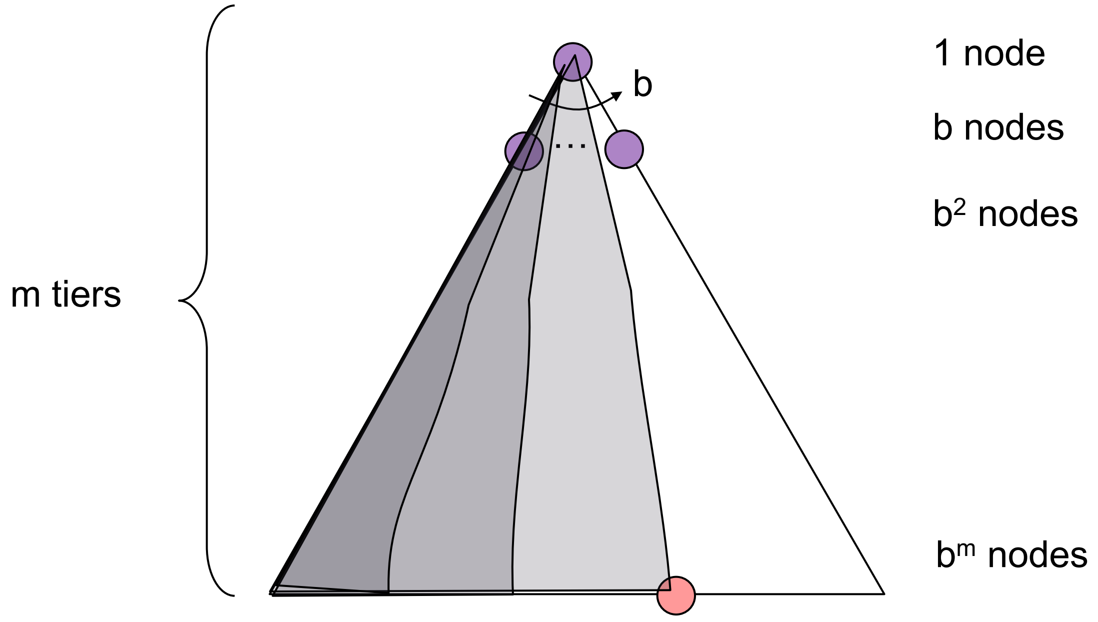
Depth-First Search (DFS)





Path?

Depth-First Search (DFS)



Depth-First Search (DFS) Properties

- Is it complete?

  If m finite state spaces, it is efficient and complete. In cyclic state spaces it can get stuck in an infinite loop.

- Is it cost optimal?

  No, it finds the “leftmost” solution, regardless of depth or cost.

- Time complexity?

  If m is finite, takes *time* $O(b^m)$ where b is the branching factor and m is the maximum depth of the tree.

- Space complexity?

  Maximum depth of tree $O(bm)$

Bidirectional search

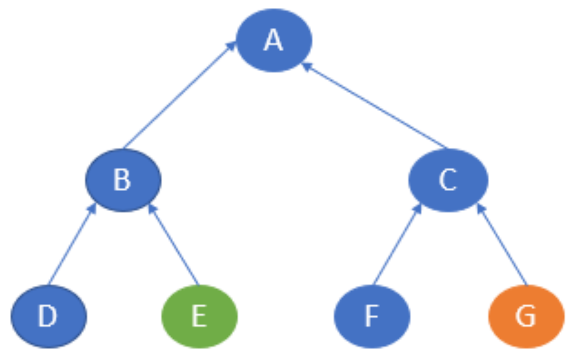
- It executes two simultaneous searches called forward-search and backwards-search and reaches the goal state.
- Here, the graph is divided into two smaller sub-graphs. In one graph, the search is started from the initial start state and in the other graph, the search is started from the goal state.
- When these two nodes intersect each other, the search will be terminated.
- **Advantages of bidirectional search**
  - This algorithm searches the graph fast.
  - It requires less memory to complete its action.
- **Disadvantages of bidirectional search**
  - The goal state should be pre-defined.
  - The graph is quite difficult to implement.

Bidirectional search (Cont.)

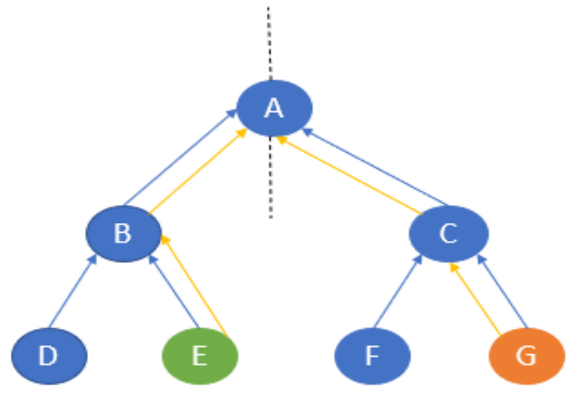
- In (a), he start state is E and the goal state is G.
- In (b), In one sub-graph, the search starts from E

and in the other, the search starts from G.

- E will go to B and then A.
- G will go to C and then A.
- Here, both the traversal meets at A and hence the traversal ends.
- The path of traversal is
  E —> B —> A —> C —> G



(a)



(b)

Bidirectional search (Cont.)

- Is it complete?

  Yes

- Is it cost optimal?

  Yes

- Time complexity?

  $O(b^{d/2})$

- Space complexity?

  $O(b^{d/2})$

Informed (Heuristic) Search Strategies

- Uses domain-specific hints about the location of goals—can find solutions more efficiently than an uninformed strategy.
- The hints come in the form of a heuristic function, denoted *h(n)*

  h(n) = estimated cost of the cheapest path from the state at node n to a goal state
- In route-finding problems, we can estimate the distance from the current state to a goal by computing the straight-line distance on the map between the two points.

Informed (Heuristic) Search Strategies

- Greedy best-first search
- A* search
- Search contours
- Satisficing search: Inadmissible heuristics and weighted A*
- Memory-bounded search
- Bidirectional heuristic search

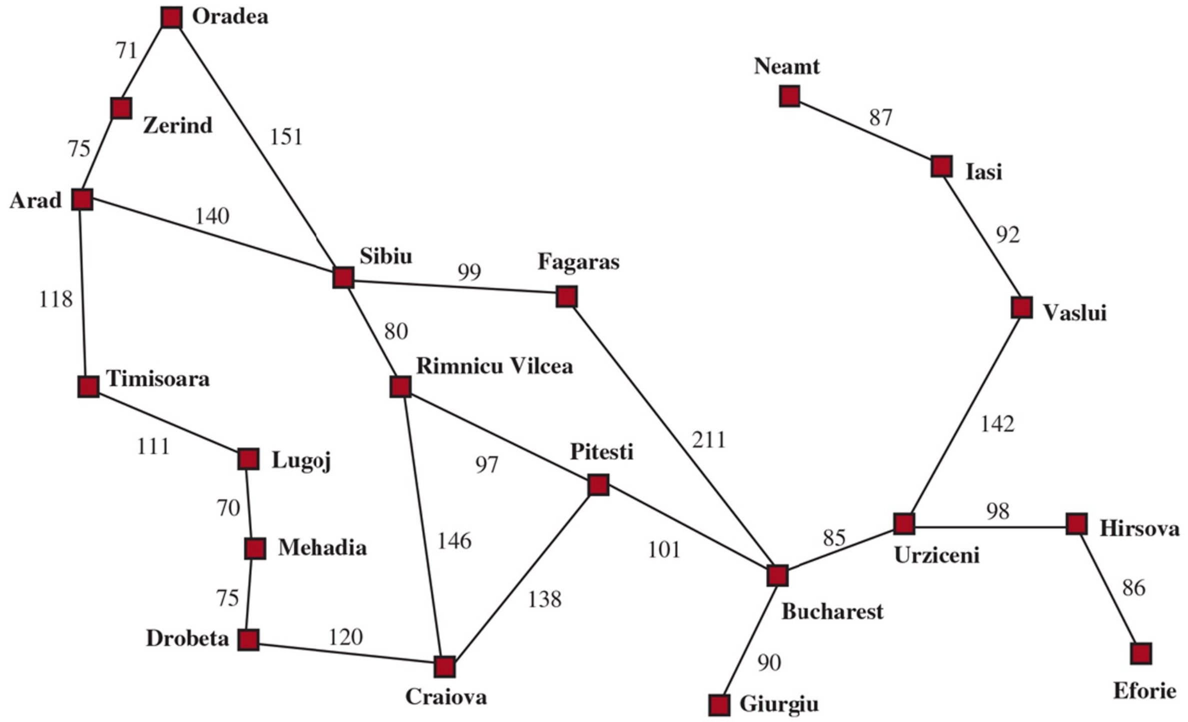
Greedy best-first search

- Greedy best-first search is a form of best-first search that expands first the node with the lowest value—the node that appears to be closest to the goal—on the grounds that this is likely to lead to a solution quickly.
- The algorithm is called "greedy", because in each step the algorithm greedily tries to get as close to the goal as possible.
- It uses only the heuristic function *h(n), s*o the evaluation function:

$$f(n) = h(n)$$

Greedy best-first search

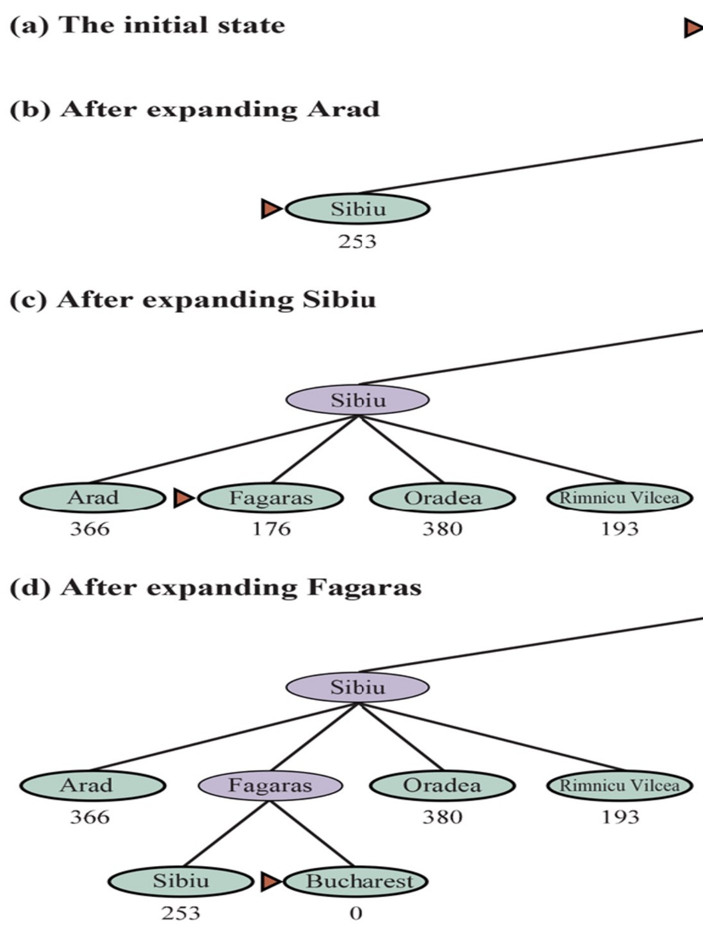
If the goal is Bucharest, we need to know h=L_{SLH} to Bucharest.



straight-line distance heuristic, h=L_{SLH}

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

Greedy best-first search



Greedy best-first search Properties

- Complete?

  Greedy best-first graph search is complete in finite state spaces, but not in infinite ones.

- Cost optimal?

  No

- Time and space complexity?

  worst-case time and space complexity is *O(|V|).* With a good heuristic function, the complexity can be reduced, on certain problems reaching *O(bm).*

A* search

- **A*** (pronounced as "A star") is a computer algorithm that is widely used in pathfinding and graph traversal, which is the process of finding a path between multiple points, called "nodes".
- It enjoys widespread use due to its performance and accuracy. However, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, although other work has found A* to be superior to other approaches.
- A best-first search that uses the evaluation function

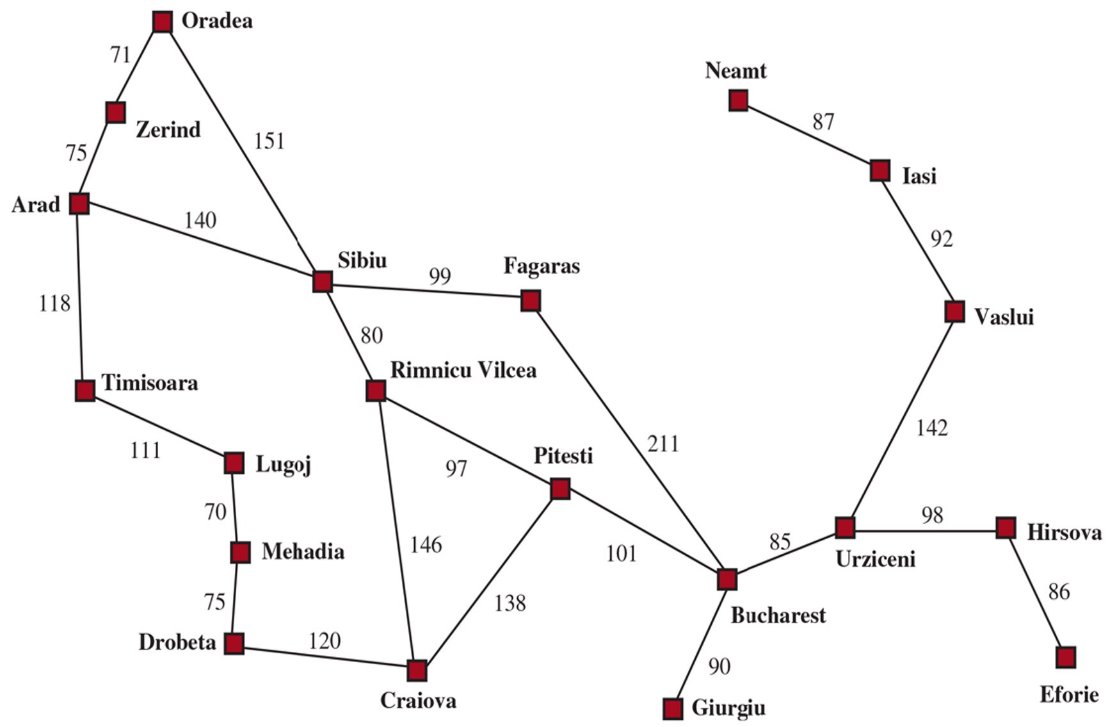
  $f(n) = g(n) + h(n)$

where $g(n)$ is the path cost from the initial state to node n, and $h(n)$ is the estimated cost of the shortest path from n to a goal state, so we have:

$f(n)$ = estimated cost of the best path that continues from n to a goal

A* search

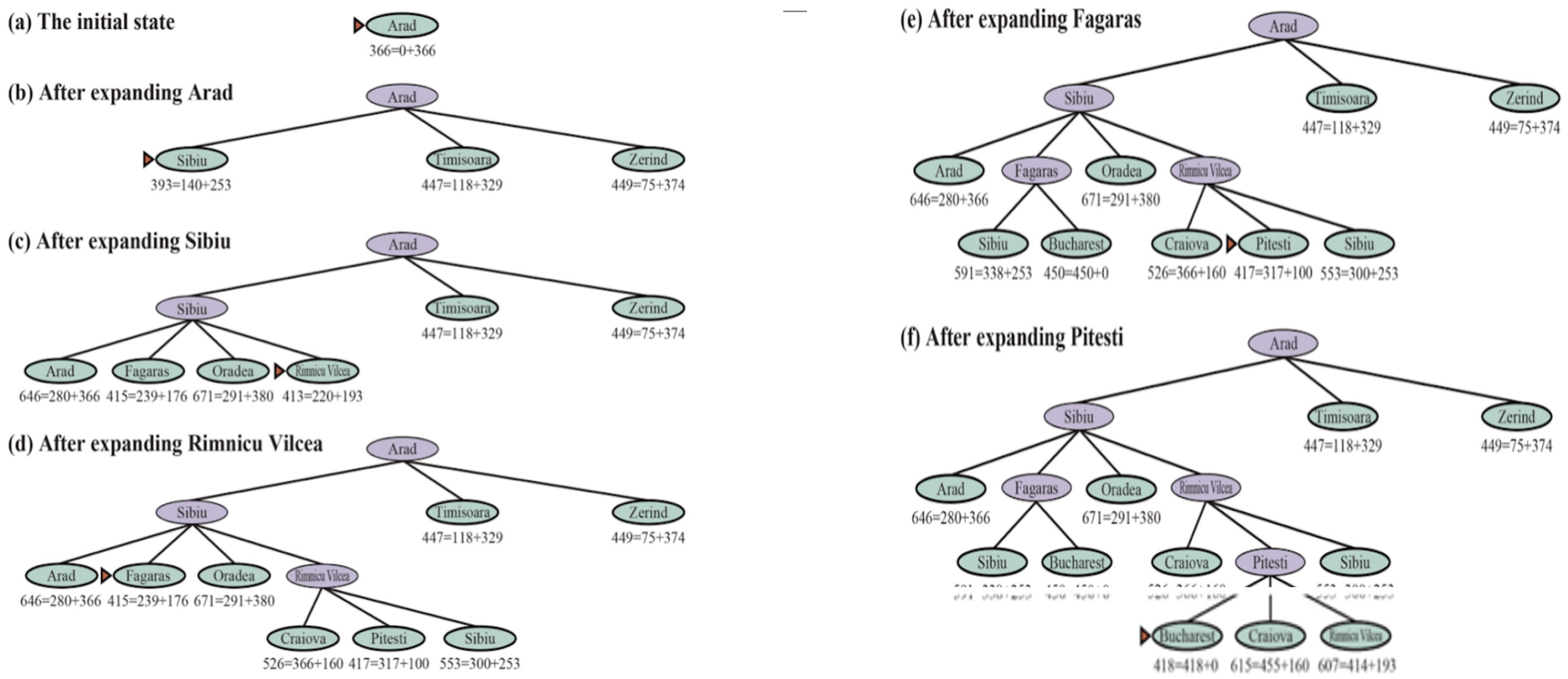
The values of *g* are computed from the action costs



h= L_{SLH}

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

A* search


(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(e) After expanding Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* search Properties

- Complete?

  Yes

- Cost optimal? 2 Key properties:
  - admissibility: an admissible heuristic is one that never overestimates the cost to reach a goal. A* is cost-optimal.
  - Consistency: a heuristic *h(n)* is consistent if, for every node *n* and every successor *n'* of *n* generated by an action *a*, we have:

    $h(n) \le c(n, a, n') + h(n')$, where *c* is cost of path,

- Time and space complexity?
  - The time and space complexity of A* depends on the heuristic where it is $O(b^d)$ in the worst case.