# Search II

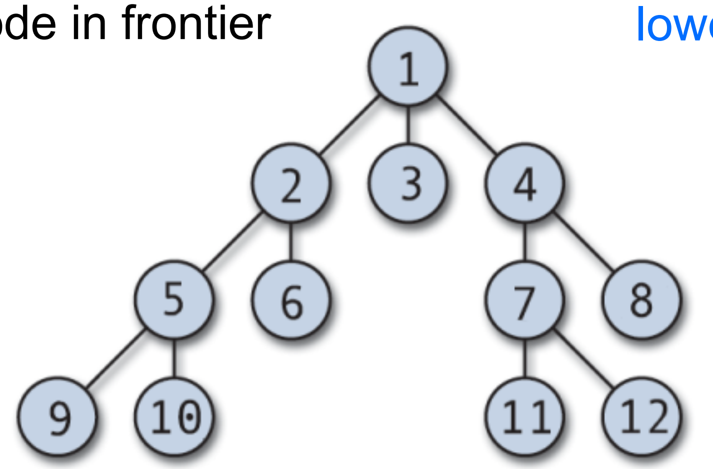
## *3. Informed Search Algorithms*

# Outline

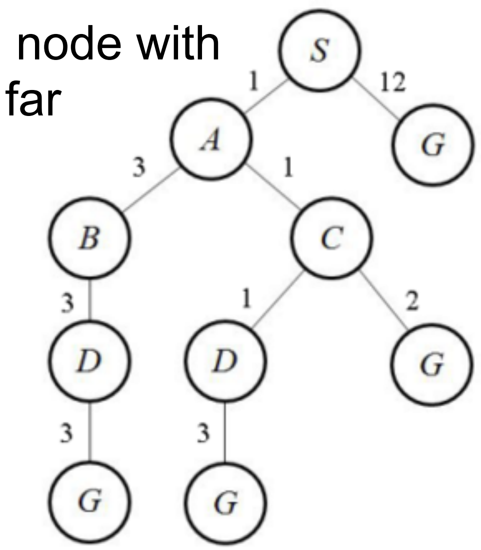
- Recap – uninformed search
- Best-first search
    - Greedy search
    - A* search
- Admissible heuristics

# Recap - Uninformed Search

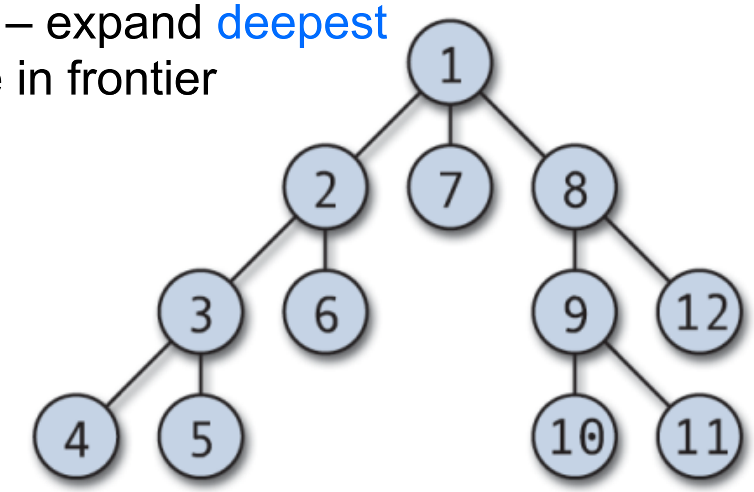
BFS – expand shallowest node in frontier



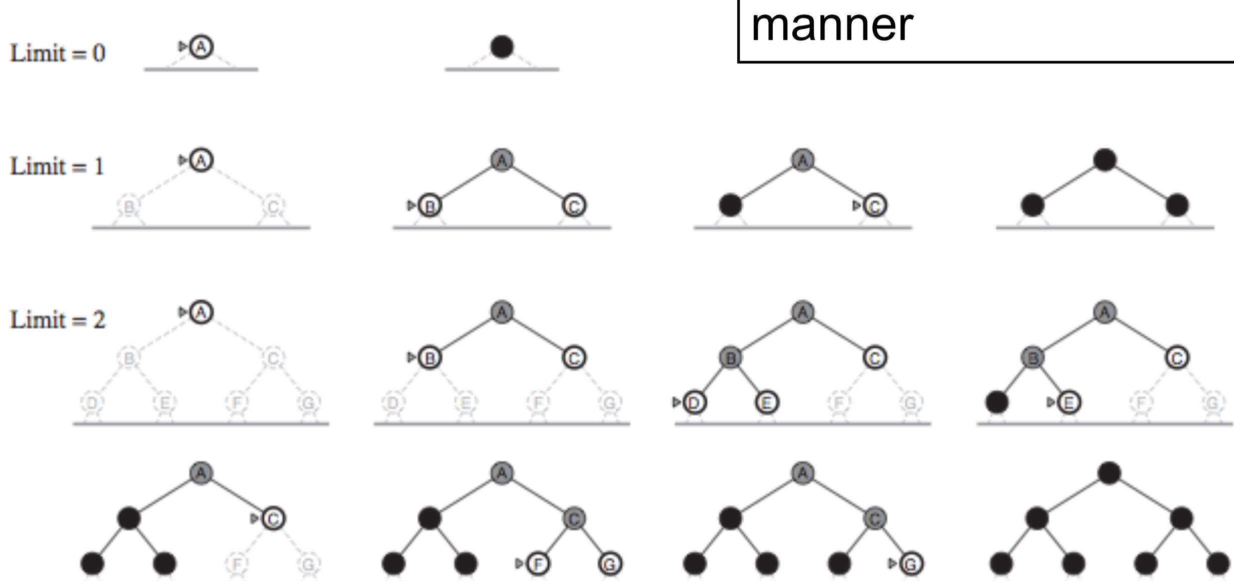

UCS – expand node with lowest cost so far

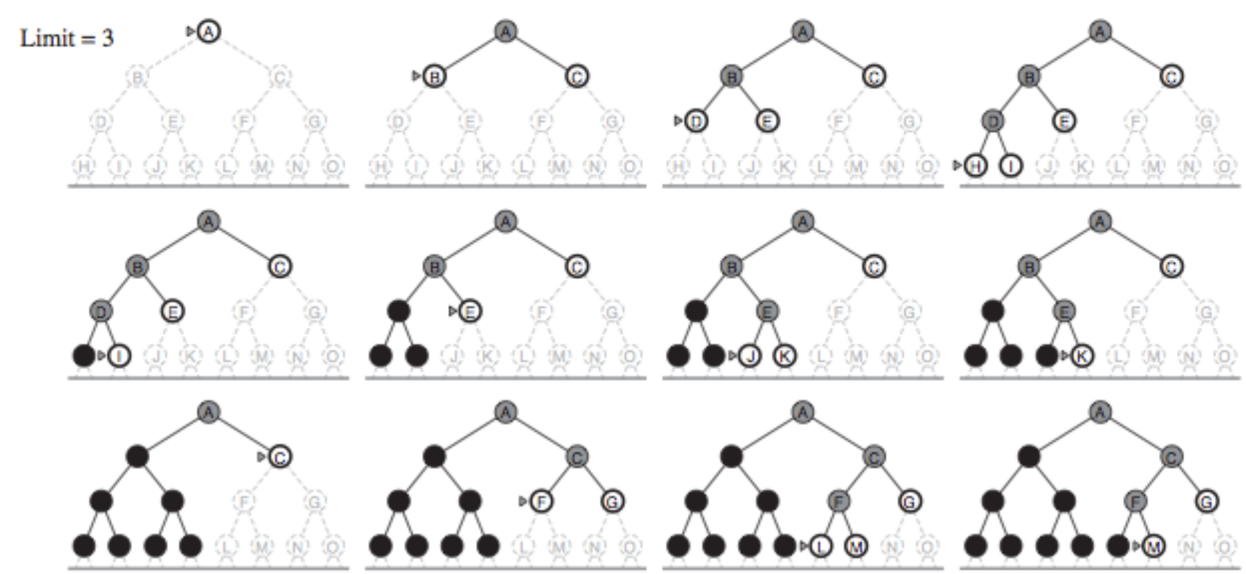



DFS – expand deepest node in frontier




IDS – DFS in BFS manner




A strategy is defined by picking the order of node expansion (based on depth level of node, or cost of getting to node (UCS)

# Informed Search Strategies

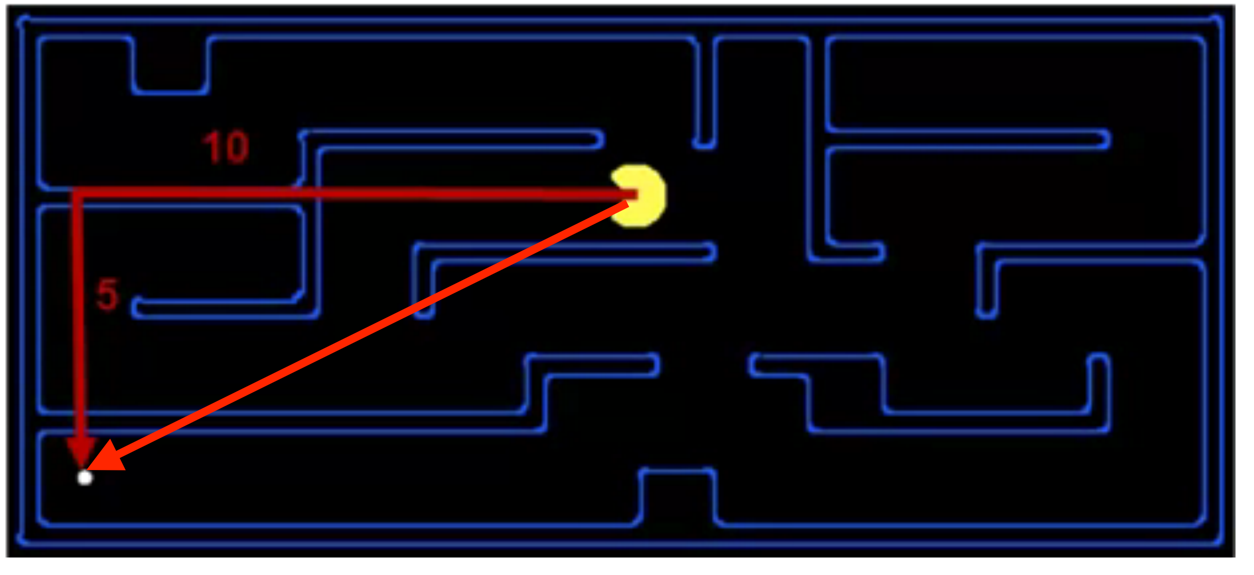
- Uninformed searches keep track of where they are now relative to where they have started (root node) in terms of step-costs
- UCS is good because it is complete and optimal but explores options in "every" direction, no information about goal location
- Need an indication if where we are right now is a good place to be for expanding nodes or a bad place to be for expanding nodes
- Use problem-specific/domain knowledge beyond the definition of the problem itself – can find solutions more efficiently than uninformed search
- Normally done using heuristics

# Heuristics

- A heuristic is a function that *estimates* how close a state is to a goal. Usually designed for a particular search problem (problem-specific)
- Most informed search algorithms include a heuristic function, *h(n)* as a component of its evaluation function*:*

  ***h(n)* = estimated cost of the cheapest path from the state at node *n* to a goal state**

- Examples are Euclidean distance (straight line distance) and Manhattan distance (good for horizontal and vertical movements) for pathing problem



# Best-first Search

- General search strategy approach for informed search is called best-first search – idea is to use an evaluation function, $f(n)$ that estimates the total cost. Node with the *lowest* $f(n)$ is expanded first (most desirable node)
- Implementation is similar to uniform-cost search, except for the use of $f$ (total cost) instead of $g$ (cost so far) to order the priority queue
- Special cases:
  - Greedy best-first search
  - A* search

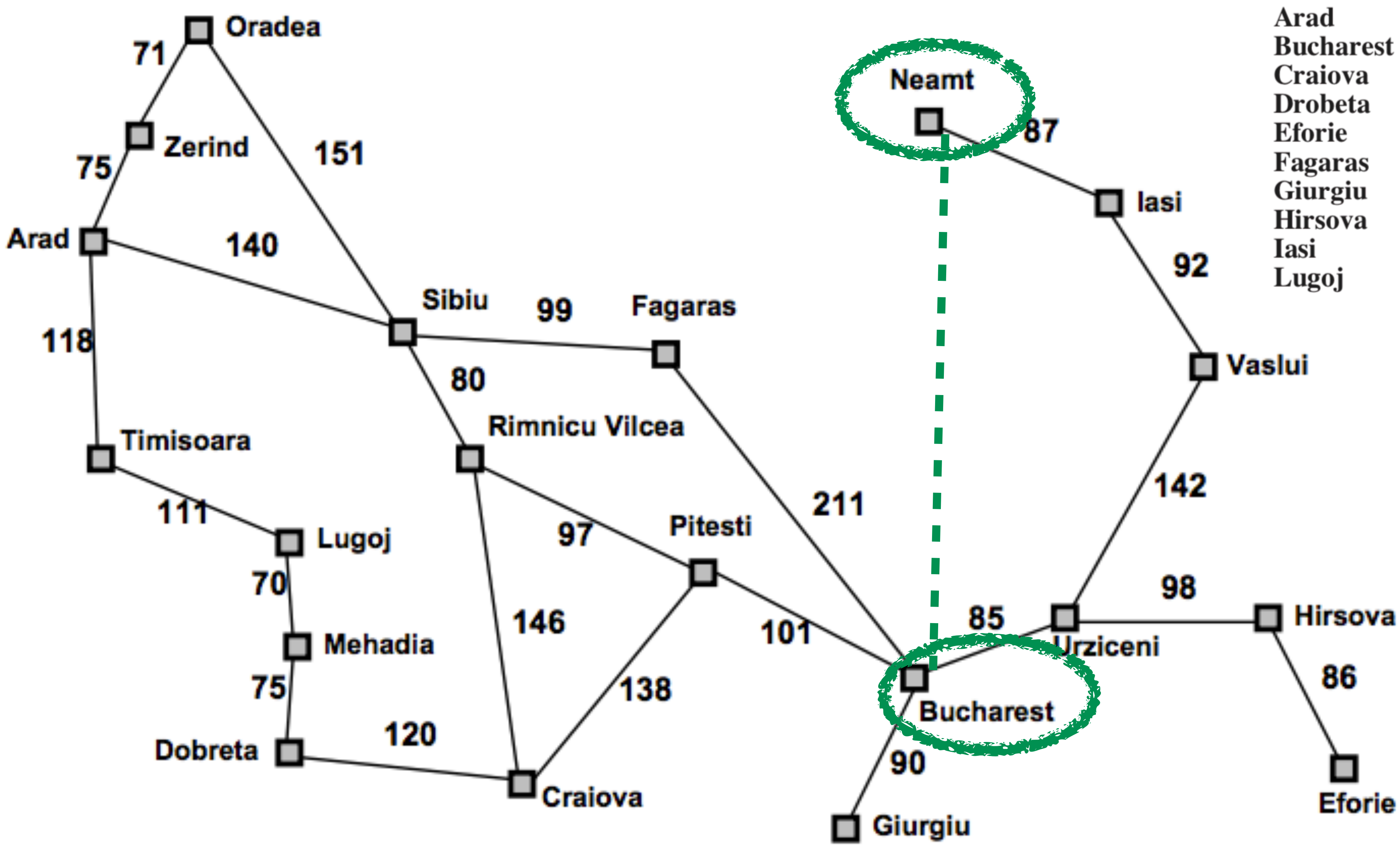
# Greedy Best-first Search

- As its name suggests, it tries to expand the node that appears to be closest to the goal in order to find the solution quickly
- Evaluates nodes using just the heuristic function:

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

- Constraint: goal node will have $h(n) = 0$

# Romania with Step Costs (km)

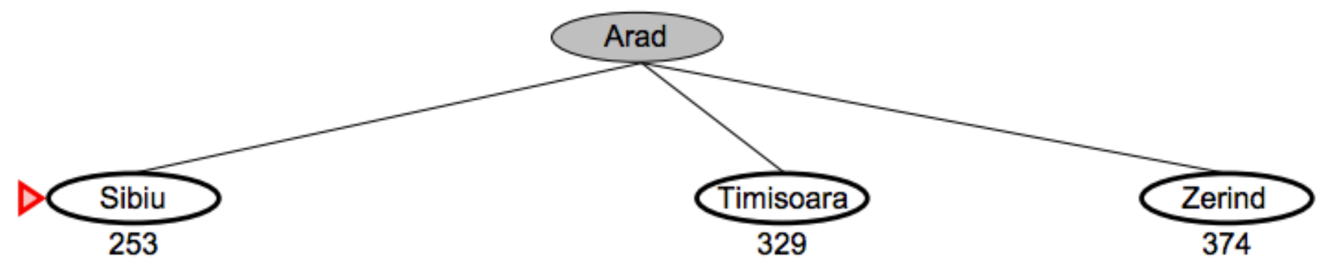


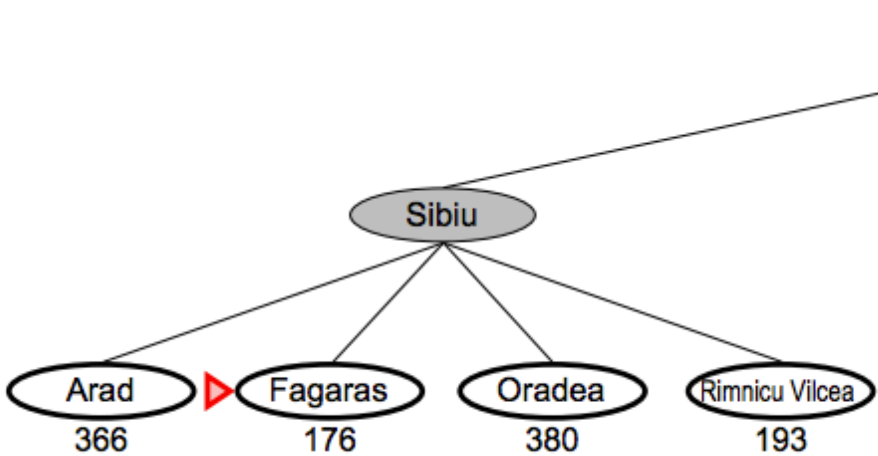

$h(n)$

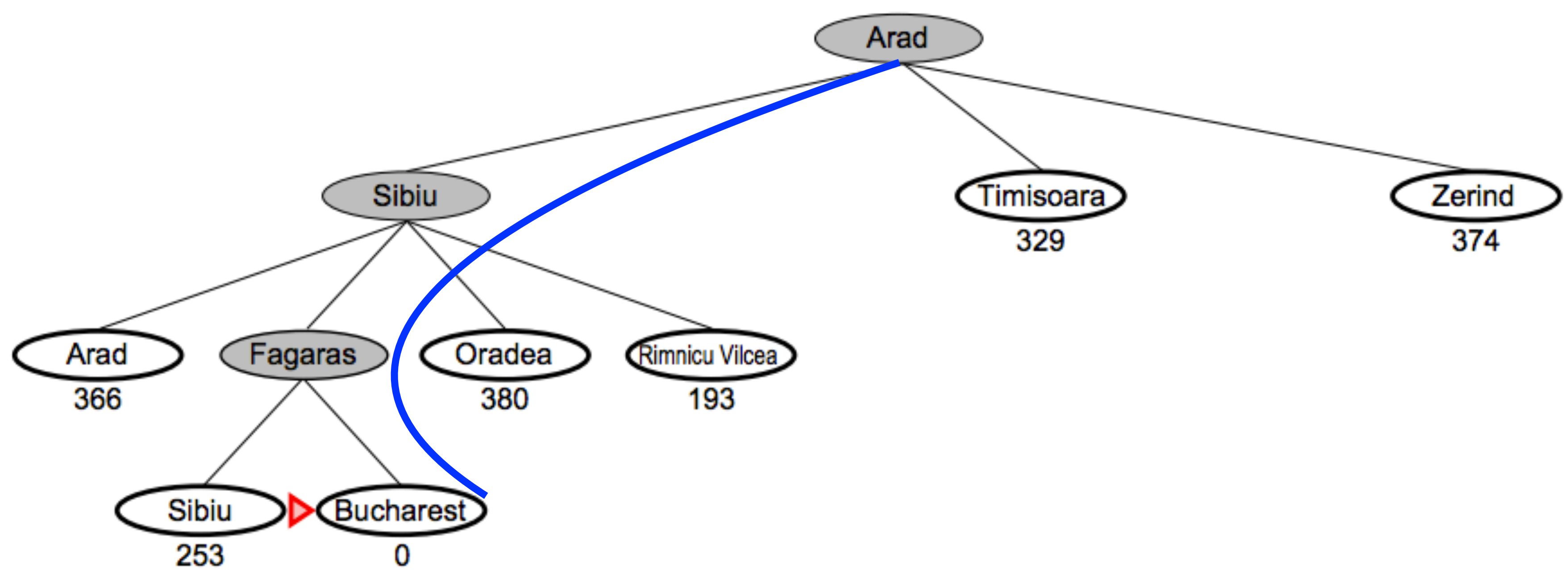
$h_{SLD}$ — straight-line distances to Bucharest

| | | | |
|---|---|---|---|
| **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 Search – Arad to Bucharest

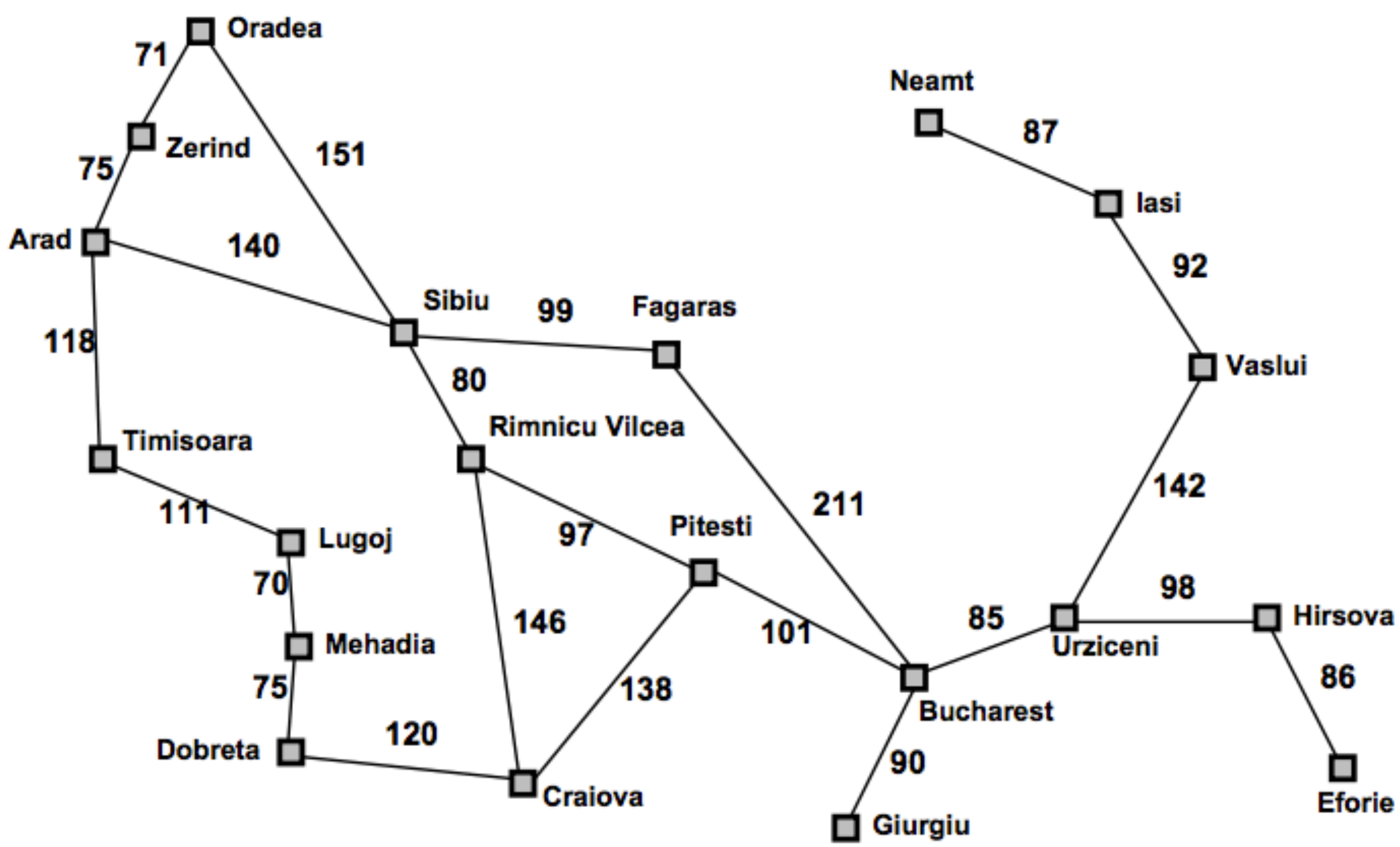




| | | | |
|---|---|---|---|
| **Arad** | 366 | **Mehadia** | 241 |
| **Bucharest** | 0 | **Neamt** | 234 |
| **Craiova** | 160 | **Oradea** | 380 |
| **Drobeta** | 242 | **Pitesti** | 100 |
| **Eforie** | 161 | **Rimnicu Vilcea** | 193 |
| **Fagaras** | 176 | **Sibiu** | 253 |
| **Giurgiu** | 77 | **Timisoara**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

# Greedy Search – Properties




Straight–line distance to Bucharest

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

- Complete? No, can get stuck in loops, e.g., *Iasi–Neamt–Iasi–Neamt– ...* (tree-search version)
- Optimal? No. In this example, *Arad–Sibiu–Fagaras–Bucharest* is 32kms longer than *Arab–Sibiu–Rimnicu Vilcea–Pitesti–Bucharest*
- Time? $O(b^m)$ but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ – keeps all nodes in memory

# A* Search

- Idea: avoid expanding paths that are already expensive
- Combines UCS with Greedy
- Evaluates nodes by combining $g(n)$, the cost so far to reach node $n$, and $h(n)$, which is the estimated cost to goal from $n$

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

  $f(n)$ = estimated cost of the cheapest solution through $n$

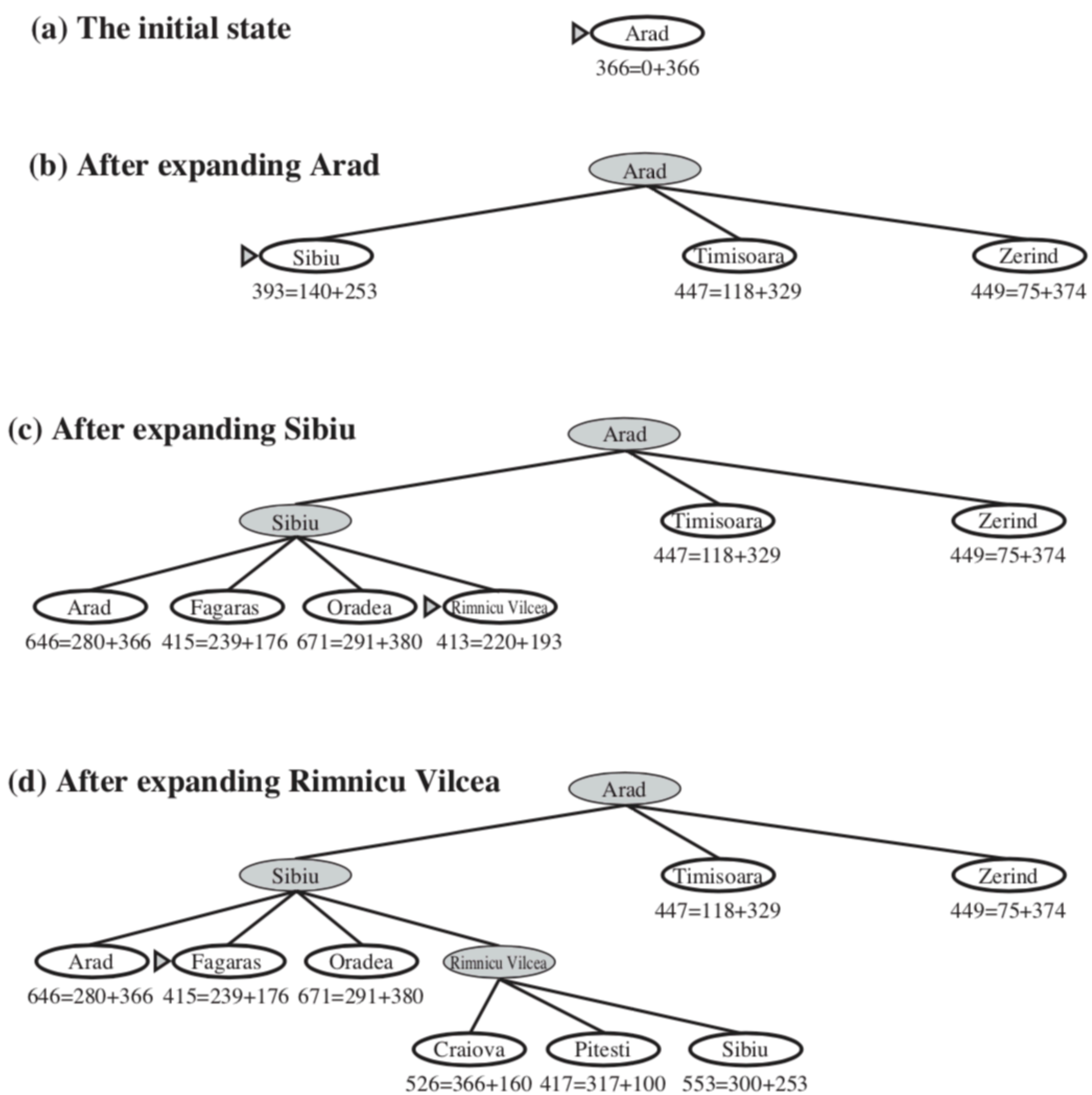
- Want $f(n)$ to be minimal
- If $h(n)$ satisfies some conditions, A* search is both complete and optimal
- Identical to UCS except that A* uses $g+h$ instead of $g$

# A* Tree Search Pseudocode

**A* tree search:** `f(n) = g(n) + h(n)`

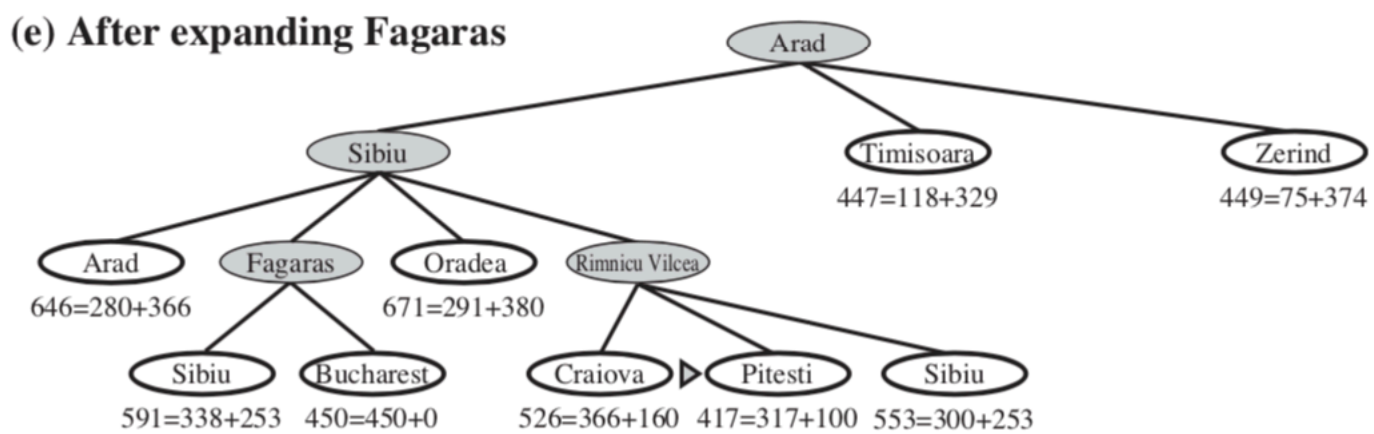
```
function A-STAR-SEARCH returns a solution or failure
    node ← INITIAL-STATE
    frontier ← a priority queue ordered by f(n) = g(n) + h(n), with node as the only element
    loop do
        if EMPTY(frontier) then return failure
        node ← POP(frontier) /* chooses the node with lowest f(n) */
        if node == Goal node then return SOLUTION(node)
        for each action in ACTIONS(node.STATE) do
            child ← CHILD-NODE(node, action)
            frontier ← INSERT(child, frontier)
```

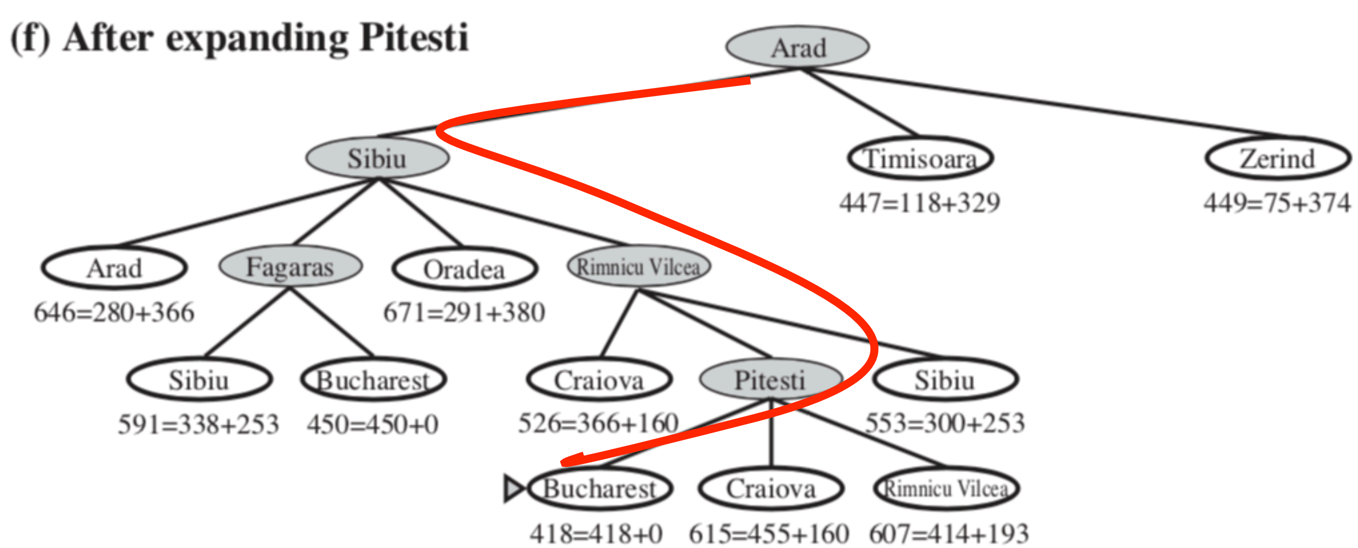
# A* Search – Arad to Bucharest




| frontier | g | h | f | |
|---|---|---|---|---|
| ~~[A]~~ | ~~0~~ | ~~366~~ | ~~**366**~~ | 1 |
| ~~[A-S]~~ | ~~140~~ | ~~253~~ | ~~**393**~~ | 2 |
| [A-T] | 118 | 329 | **447** | |
| [A-Z] | 75 | 374 | **449** | |
| [A-S-A] | 280 | 366 | **646** | |
| ~~[A-S-F]~~ | ~~239~~ | ~~176~~ | ~~**415**~~ | 4 |
| [A-S-O] | 291 | 380 | **671** | |
| ~~[A-S-RV]~~ | ~~220~~ | ~~193~~ | ~~**413**~~ | 3 |
| [A-S-RV-C] | 366 | 160 | **526** | |
| [A-S-RV-P] | 317 | 100 | **417** | |
| [A-S-RV-S] | 300 | 253 | **553** | |

# A* Search – Arad to Bucharest

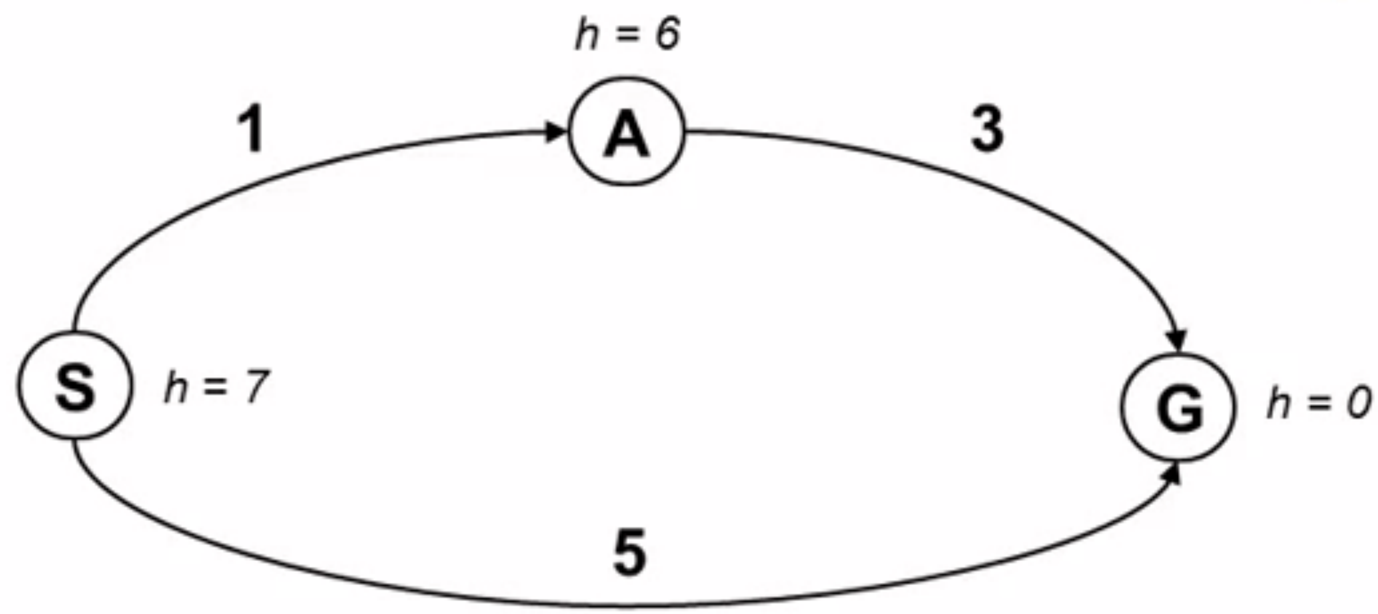


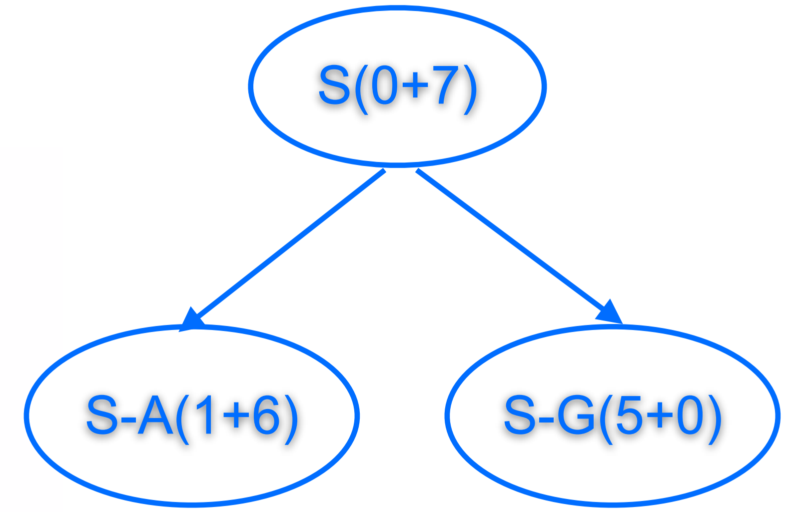




| frontier | g | h | f | |
|---|---|---|---|---|
| [A-T] | 118 | 329 | **447** | |
| [A-Z] | 75 | 374 | **449** | |
| [A-S-A] | 280 | 366 | **646** | |
| ~~[A-S-F]~~ | ~~239~~ | ~~176~~ | ~~**415**~~ | 4 |
| [A-S-O] | 291 | 380 | **671** | |
| [A-S-RV-C] | 366 | 160 | **526** | |
| ~~[A-S-RV-P]~~ | ~~317~~ | ~~100~~ | ~~**417**~~ | 5 |
| [A-S-RV-S] | 300 | 253 | **553** | |
| [A-S-F-S] | 338 | 253 | **591** | |
| [A-S-F-B] | 450 | 0 | **450** | |
| ~~[A-S-RV-P-B]~~ | ~~418~~ | ~~0~~ | ~~**418**~~ | 6 |
| [A-S-RV-P-C] | 455 | 160 | **615** | |
| [A-S-RV-P-RV] | 414 | 193 | **607** | |

Since **[A-S-RV-P-B-418]** is the cheapest path in the frontier and passes the goal test, return it as the SOLUTION

# When is A* Optimal?




S-G-5 passes the goal test
and returned as the SOLUTION

- A* returned S-G (actual cost 5) although S-A-G (actual cost 4) is the optimal solution
- Why?
  - Because A's heuristics is a bad estimate, it is much higher than the real cost of getting from A to G (which is 3)
- We need estimates (heuristics) to be less than actual costs!

estimated cost to goal $h(n)$ ≤ actual cost to goal $h^*(n)$

# Conditions for Optimality: Admissibility

- For A* search to be optimal, it needs an admissible heuristic, i.e.

  $0 \leq h(n) \leq h^*(n)$

  where $h^*(n)$ is the true cost of the cheapest solution from $n$ to a nearest goal

- An admissible heuristic *never overestimates* the cost to reach the goal

- Since $g(n)$ is the actual cost to reach $n$ along the current path, and $f(n) = g(n) + h(n)$, we can conclude that $f(n)$ never overestimates the true cost of a solution along the current path through $n$

* Why is the straight-line distance, $h_{SLD}(n)$ an admissible heuristic for Romania?

* What about the Manhattan distance for Pacman getting to a location in the maze?

# A* Properties

- Complete? Yes, unless there are infinitely many nodes with $f < f(G)$
- Optimal? Yes – cannot expand $f_{i+1}$ until $f_i$ is finished
- Time? Exponential in [relative error of $h$ x length of solution]
- Space? Keeps all nodes in memory. A* usually runs out of space long before it runs out of time ∴ not practical for many large-scale problems
- If C* is the cost of the optimal solution path:

A* expands all nodes with $f(n) < C^*$

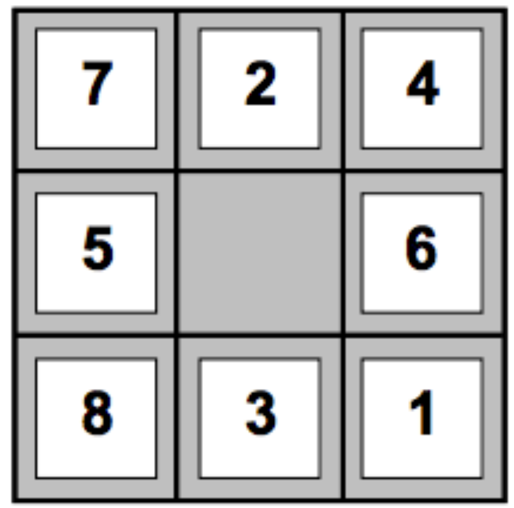
A* expands some nodes with $f(n) = C^*$

A* expands no nodes with $f(n) > C^*$

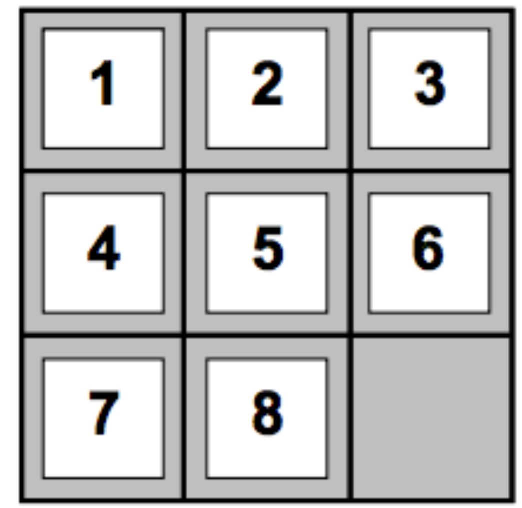
# A* Applications

- Pathing problems
- Video games
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition

# Heuristics for 8-puzzle




**Start State**

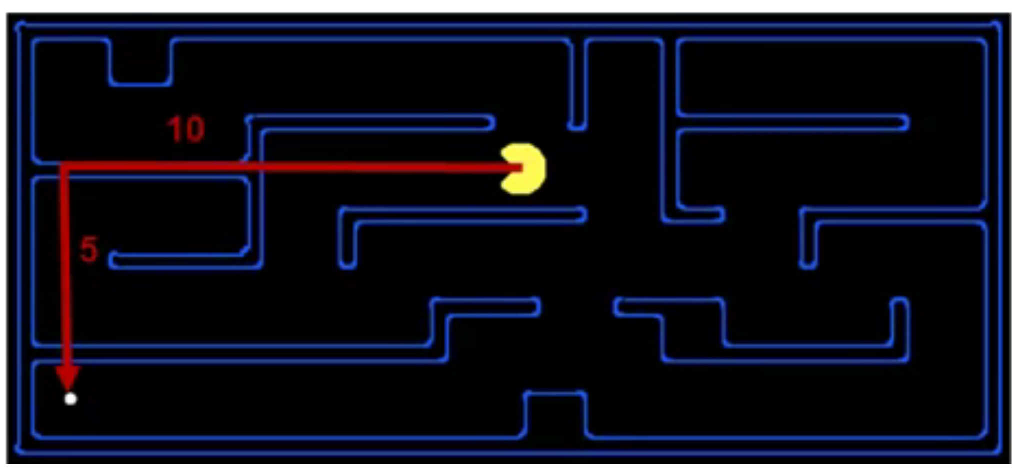



**Goal State**

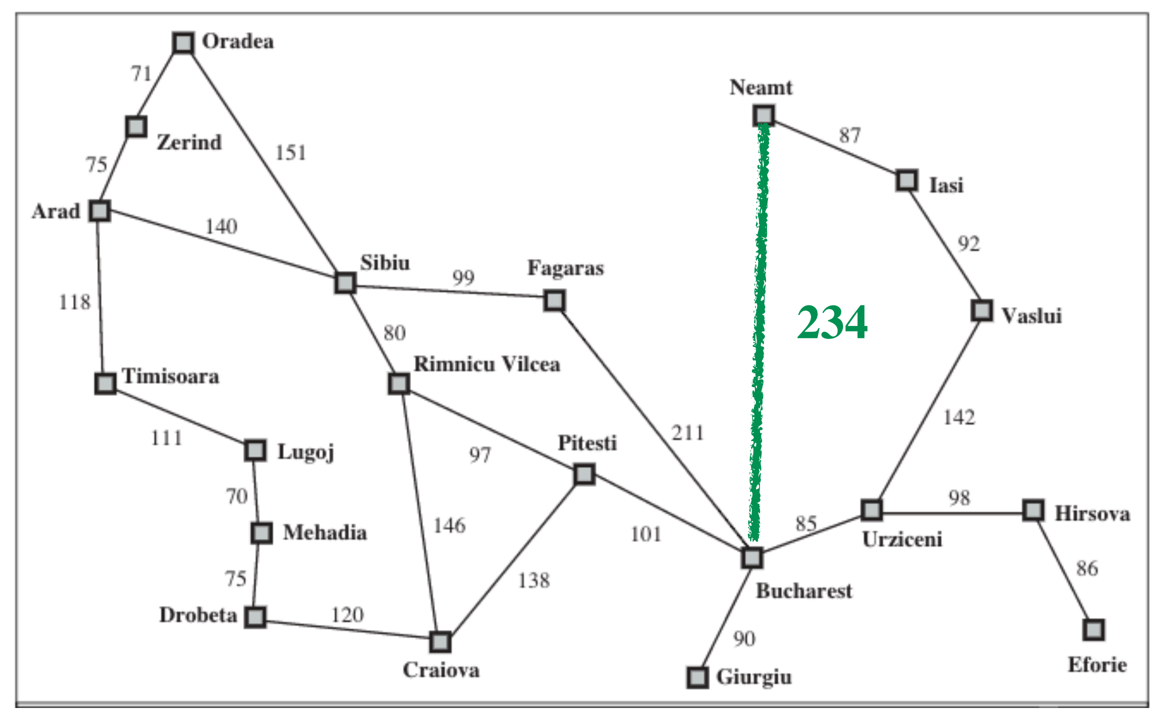
- Slide tiles horizontally or vertically into the empty space until the configuration matches the goal configuration
- Branching factor = number of possible moves at a particular state (maximum = 4, average = 3)
- Want to find shortest solutions by using A*:
  - $h_1$ = number of misplaced tiles
  - $h_2$ = total Manhattan distance (no. of squares from desired location of each tile or the sum of the distances of the tiles from their goal positions)
- Q: What are $h_1$ and $h_2$ for the Start State above?

# Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is coming up with admissible heuristics
- Often admissible heuristics are solutions to relaxed problems where new actions are available




Pacman relaxed version: No wall

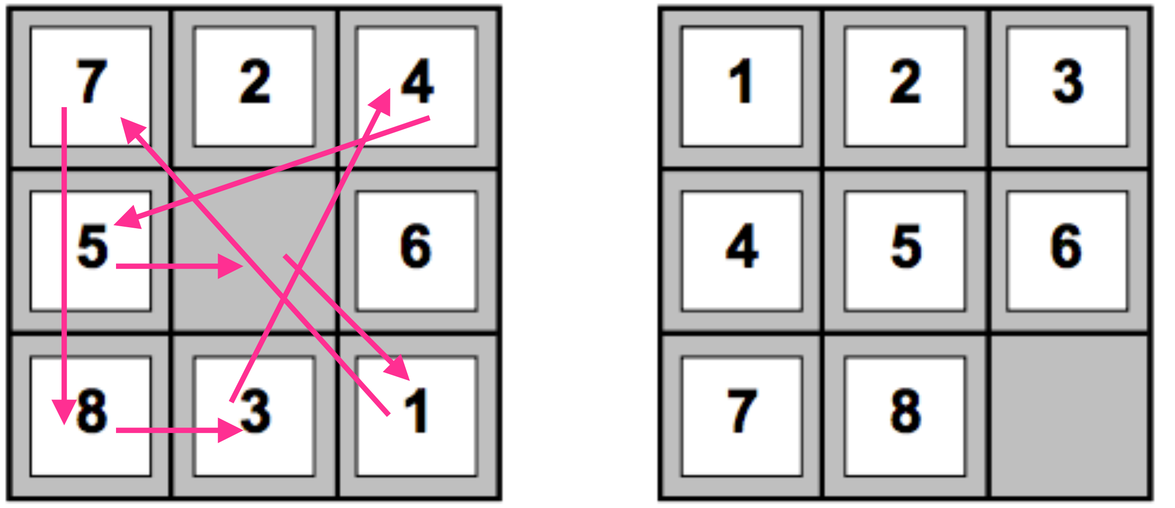


# Relaxed Problems

- In 8-puzzle, what if the rules were relaxed so that a tile could move anywhere instead of just to an adjacent square?

  $h_1(n)$ (number of misplaced tiles) gives the shortest solution

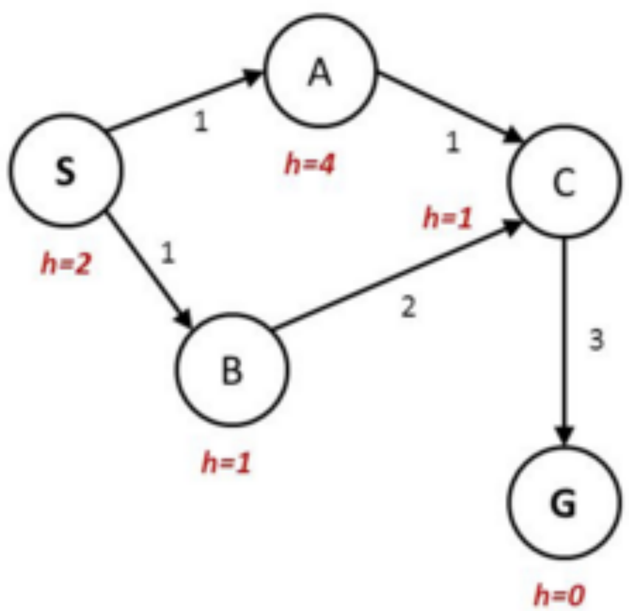
- If the rules were relaxed so that a tile could move to any adjacent square, then $h_2(n)$ (Manhattan distance) gives the shortest solution
- A relaxed problem is one with fewer restrictions on the actions (e.g. *simplified* version of the puzzle)
- $h(n)$ can be generated using machine learning techniques



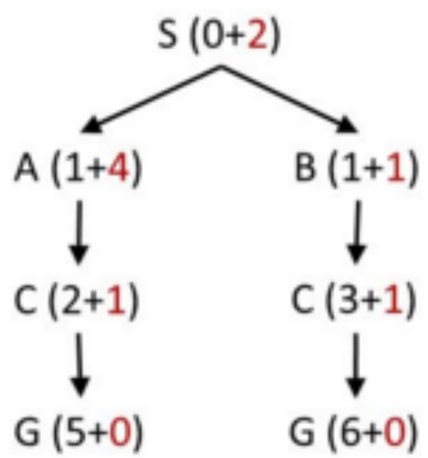

**Start State** **Goal State**

# A* Graph Search

State space graph

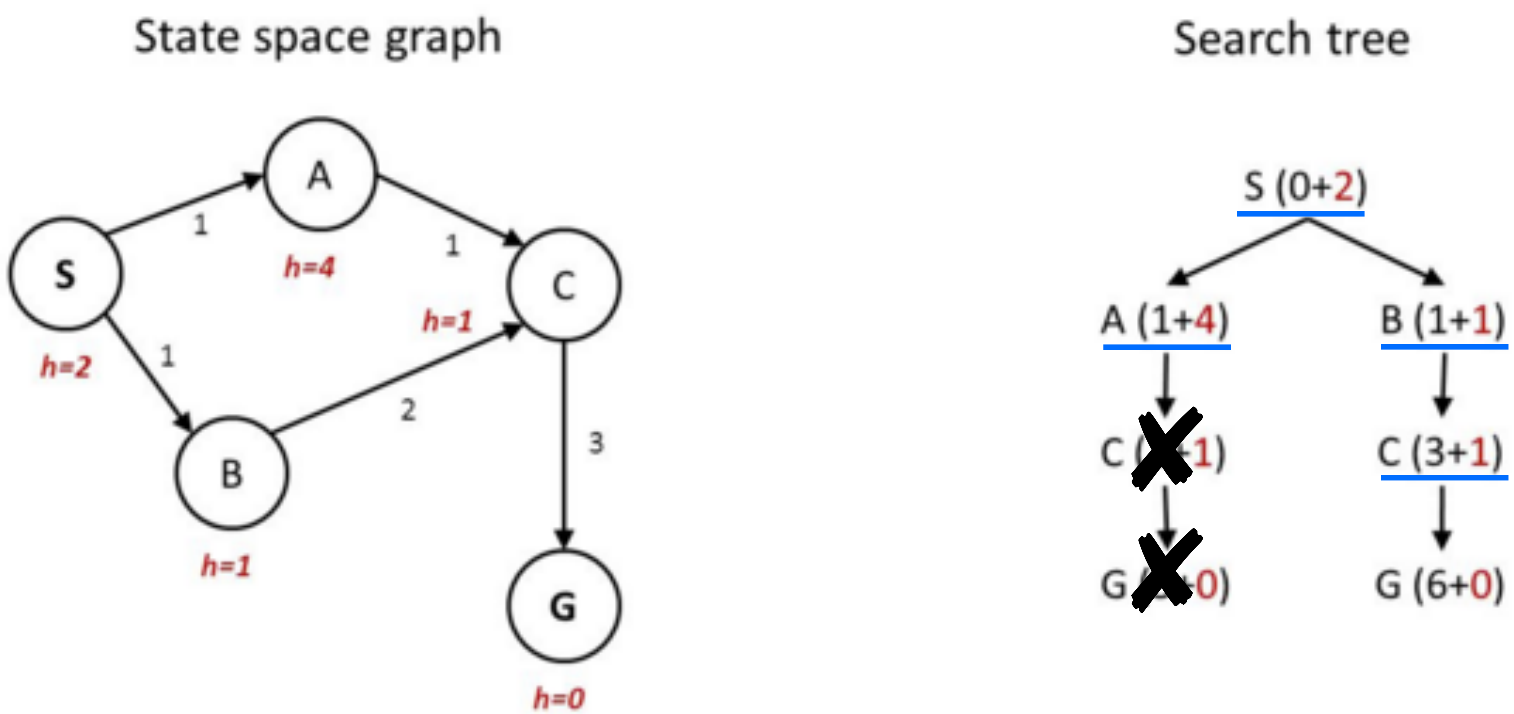



Search tree




Q: What went wrong here?

# A* Graph Search




Q: What went wrong here?

A* graph search is not optimal because it doesn't expand C(2+1) as it is already in the explored list via C(3+1), so it fails to return the optimal solution S-A-C-G

# Summary

- Informed search strategies may have access to heuristic functions $h(n)$ that estimate the cost of the shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands the lowest $h$
  - incomplete and not always optimal
- A* expands lowest $g+h$ (UCS and Greedy)
  - complete and optimal (and optimally efficient)
- Admissible heuristics never overestimate the cost to the goal
- Heuristic design is key: can be constructed by relaxing the problem (e.g. 8-puzzle)

# References

- Russel and Norvig, Chapter 3.5–3.6
- Greedy search quick demo [Link]
- A* search demo [Link]
- A* search explanation by John Levine [Link]
- Code in C++ for BFS, DFS, UCS, Greedy and A*, Y.Li [Link]