EECS 348: Informed Search

Tree Search Algorithm

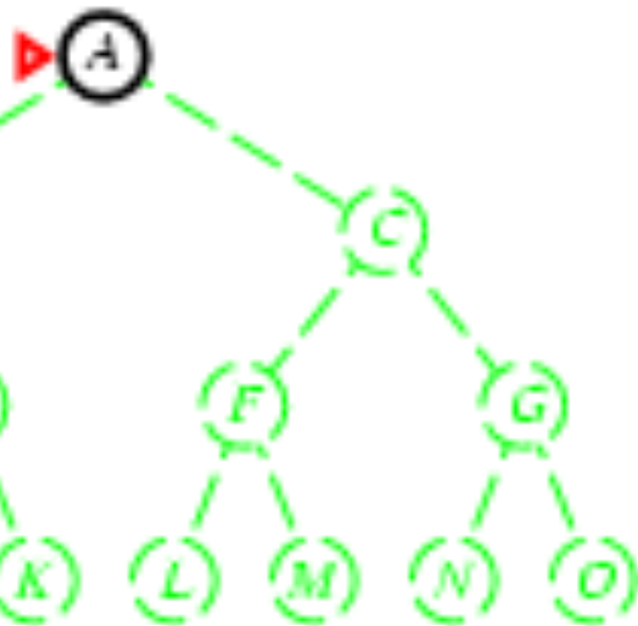
1. Add the initial state (root) to the <fringe>
2. **Choose a node (curr) to examine from the <**fringe**>** (if there is nothing in <fringe> - FAILURE)
3. Is curr a goal state?
   - If so, SOLUTION
   - If not, continue
4. Expand curr by applying all possible actions (add the new resulting states to the <fringe>)
5. Go to step 2

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-limited search

= depth-first search with depth limit L,

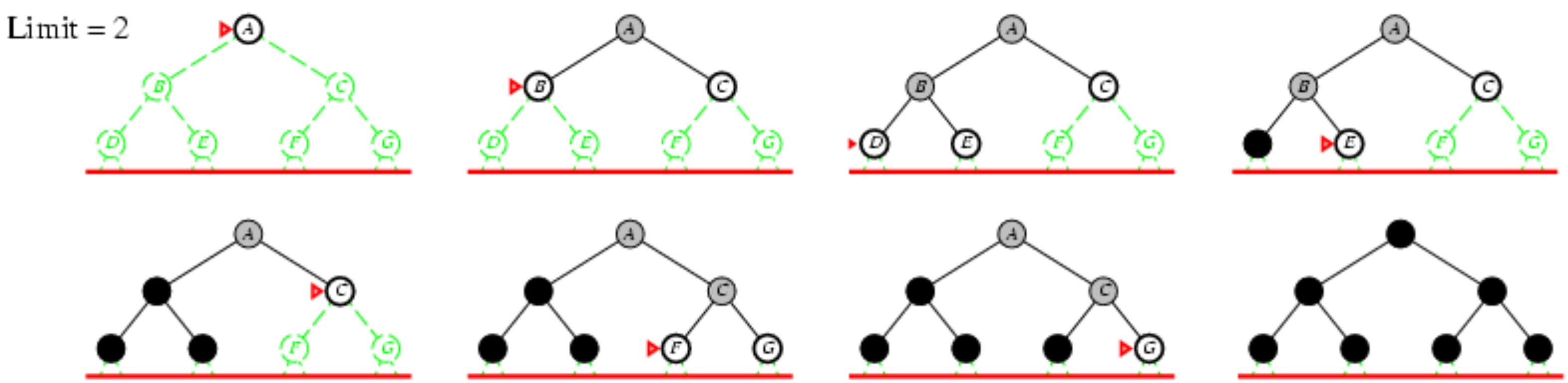
i.e., nodes at depth L have no successors

Iterative deepening search *L* =0

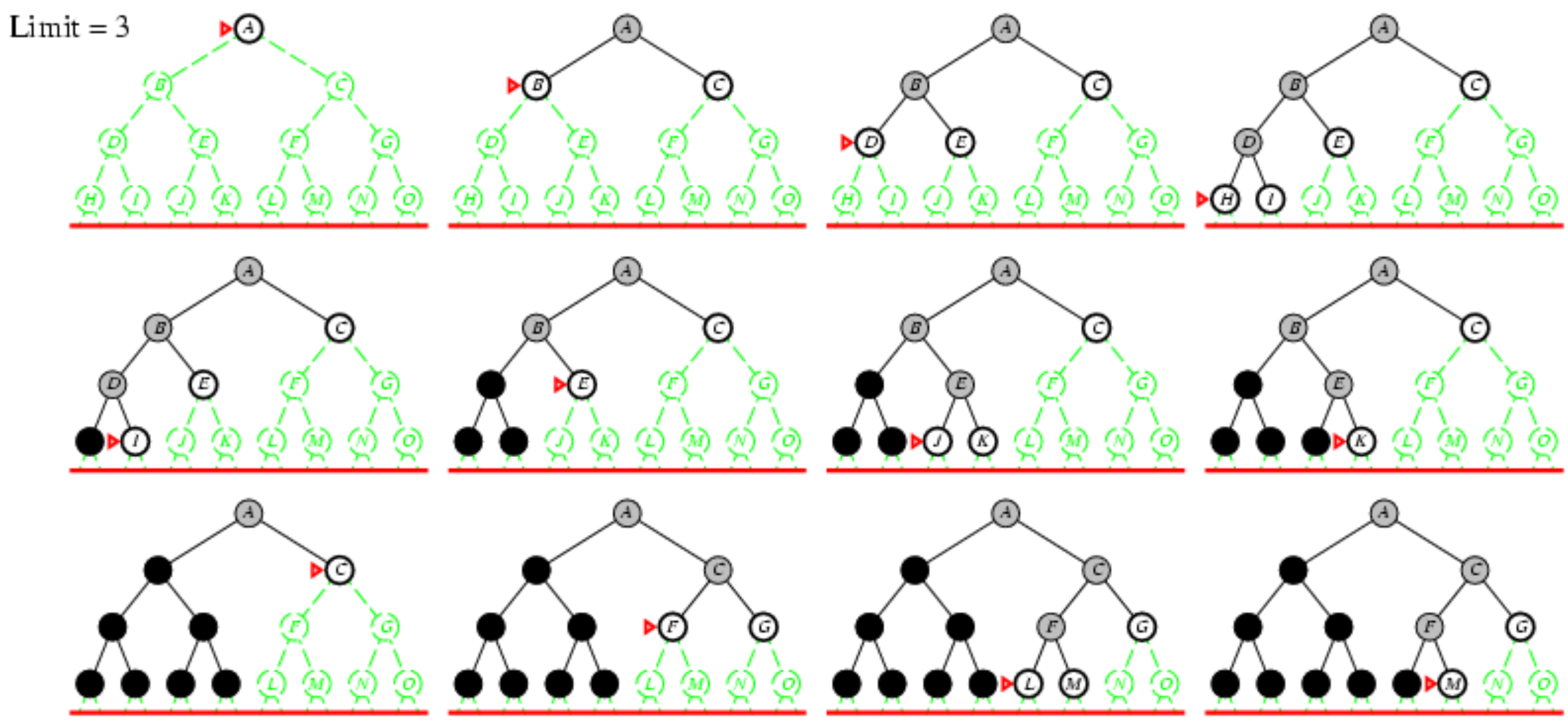
Limit = 0

Iterative deepening search L =1

Iterative deepening search L =2



Iterative deepening search L =3



Properties of iterative deepening search

- Space
  - *O(bd)*
- Complete?
  - Yes
- Optimal?
  - Yes

Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

Time?

- L = 0: 1
- L = 1: $1 + b$
- L = 2: $1 + b + b^2$
- L = 3: $1 + b + b^2 + b^3$
- …
- L = d: $1 + b + b^2 + b^3 + \ldots + b^d$
- Overall:
  - $(d+1)b^0 + (d)b^1 + (d-1)b^2 + (d-2)b^3 + \ldots 2b^{d-1} + b^d$
  - $O(b^d)$
  - the cost of the repeat of the lower levels is subsumed by the cost at the highest level

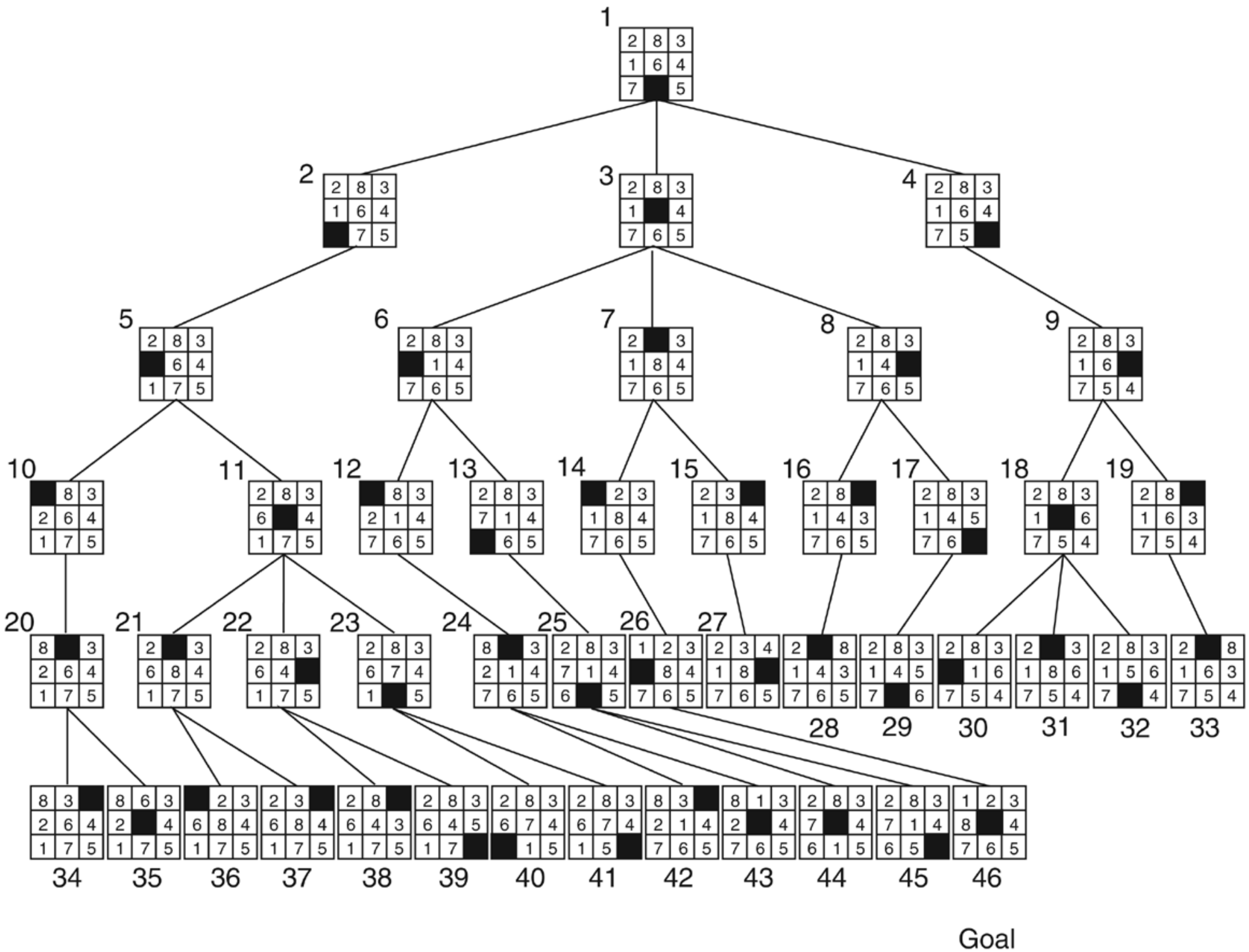
Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes

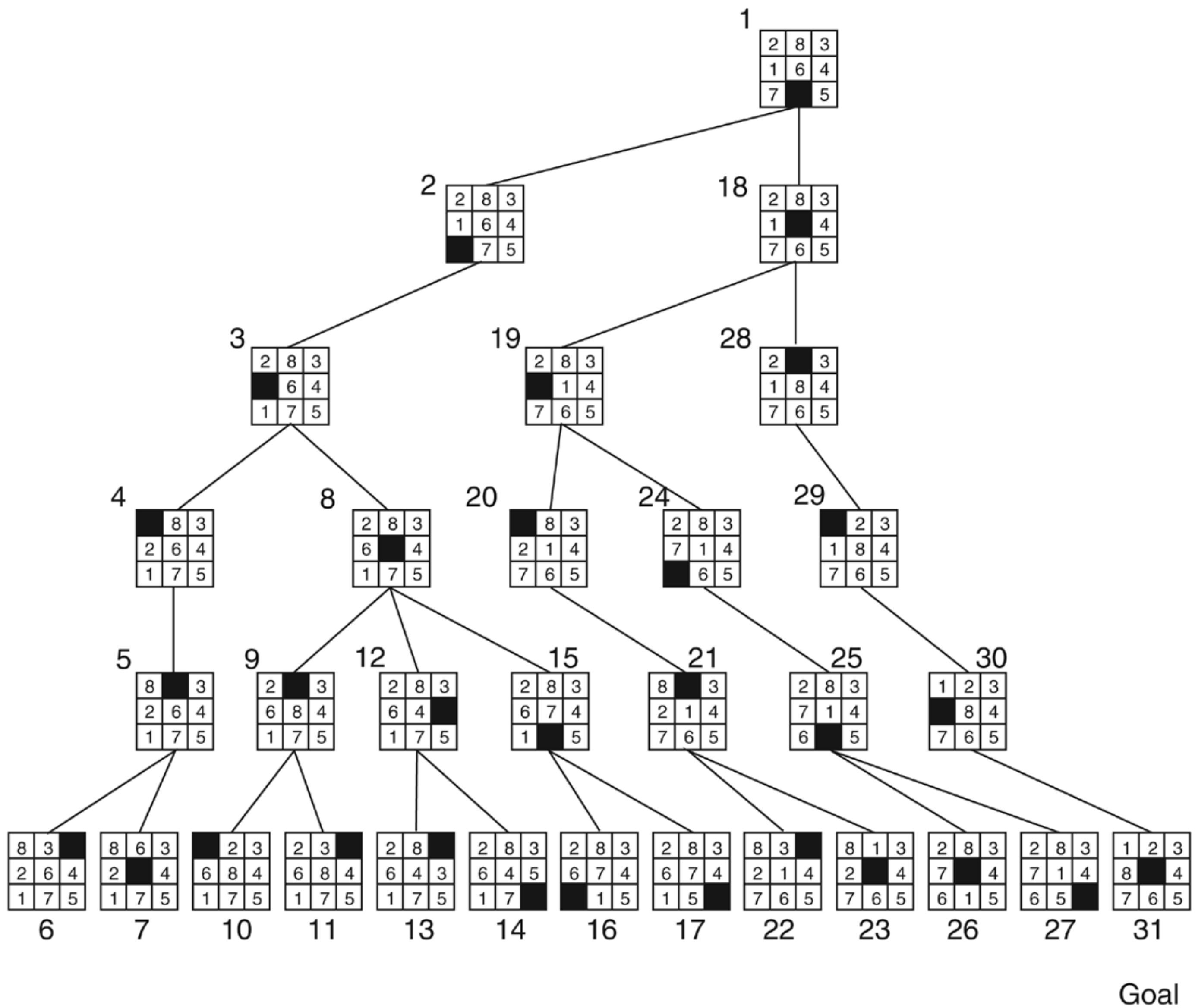
Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Breadth-first search of the 8-puzzle - # of node denotes order in which the nodes are visited.



Depth-first search of the 8-puzzle with a depth bound of 5 (# is the order the nodes are examined - pic is missing nodes that were added but not examined).



Tree Search Algorithm

1. Add the initial state (root) to the <fringe>
2. **Choose a node (curr) to examine from the** **<**fringe**>** (if there is nothing in <fringe> - FAILURE)
3. Is curr a goal state?
    - If so, SOLUTION
    - If not, continue
4. Expand curr by applying all possible actions (add the new resulting states to the <fringe>)
5. Go to step 2

Two heuristics applied to states in the 8-puzzle.

State	Tiles out of place	Sum of distances out of place
2 8 3 1 6 4 7 5	**5**	**6**
2 8 3 1 4 7 6 5	**3**	**4**
2 8 3 1 6 4 7 5	**5**	**6**

1	2	3
8		4
7	6	5

Goal

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"

  → Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)

  = estimate of cost from n to *goal*

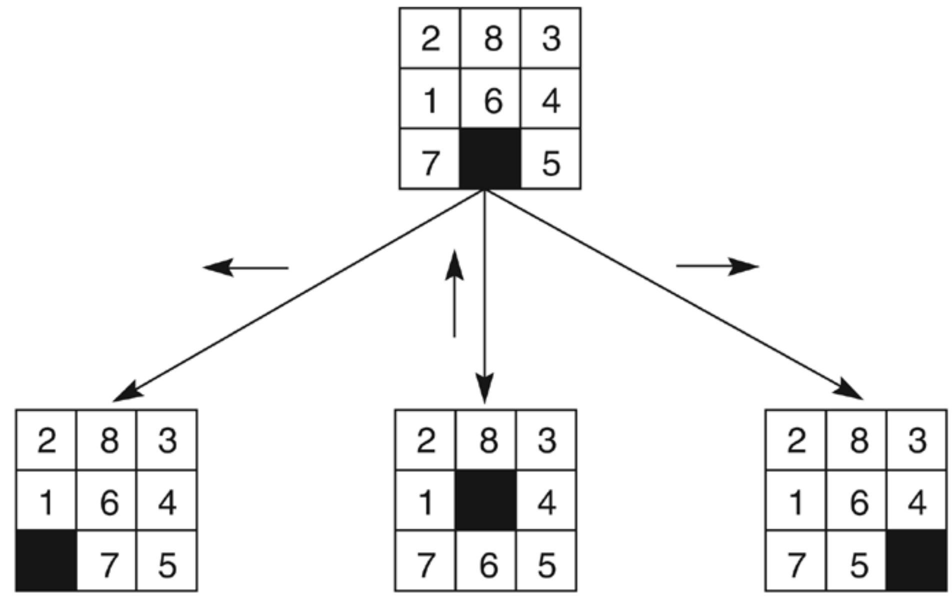
A^* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal (the evaluation of the desirability of n)

The heuristic **f** applied to states in the 8-puzzle.

Start

g(n) = 0

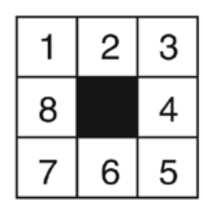


g(n) = 1

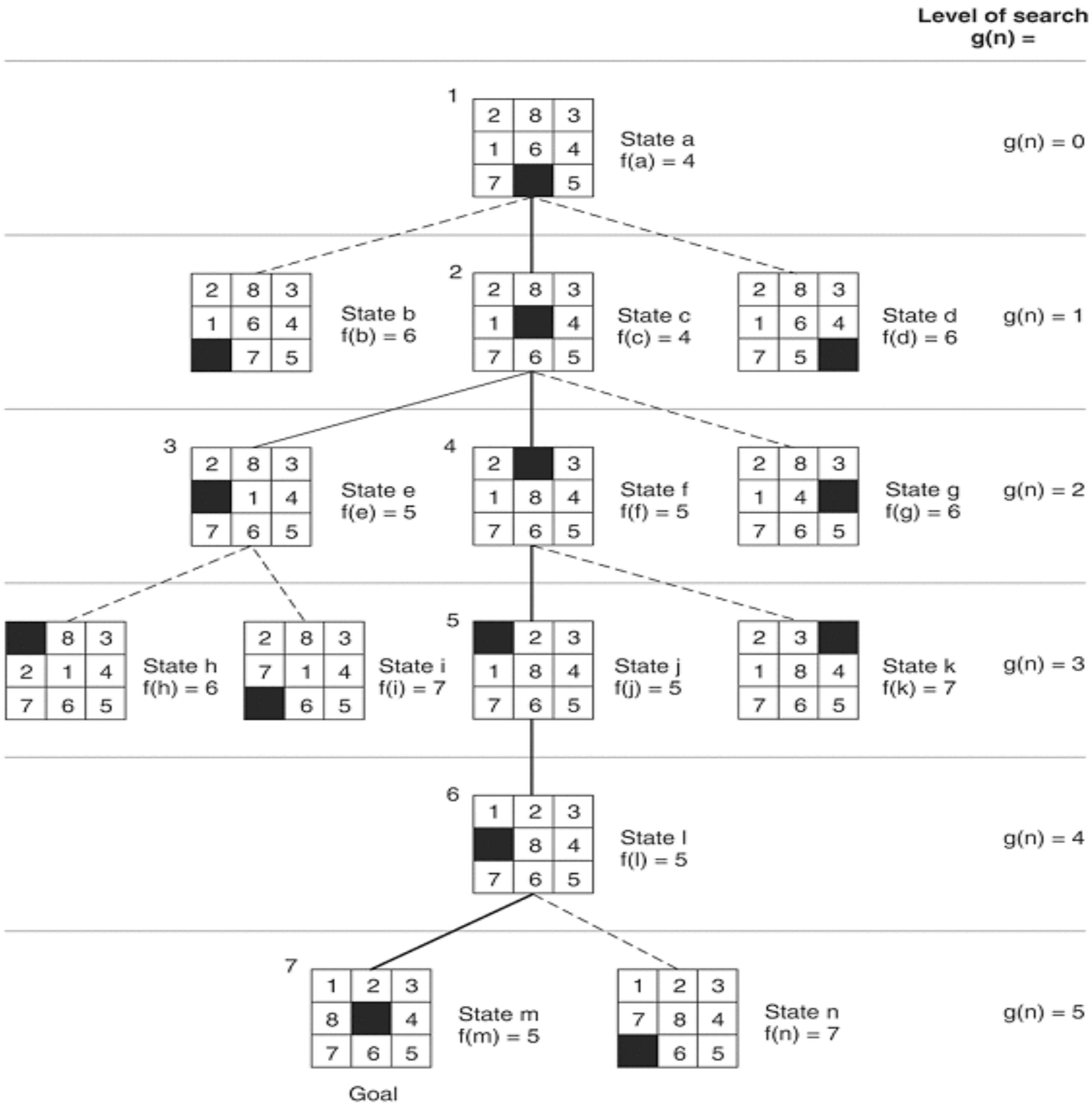
Values of f(n) for each state, **6** **4** **6**

where:

f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.



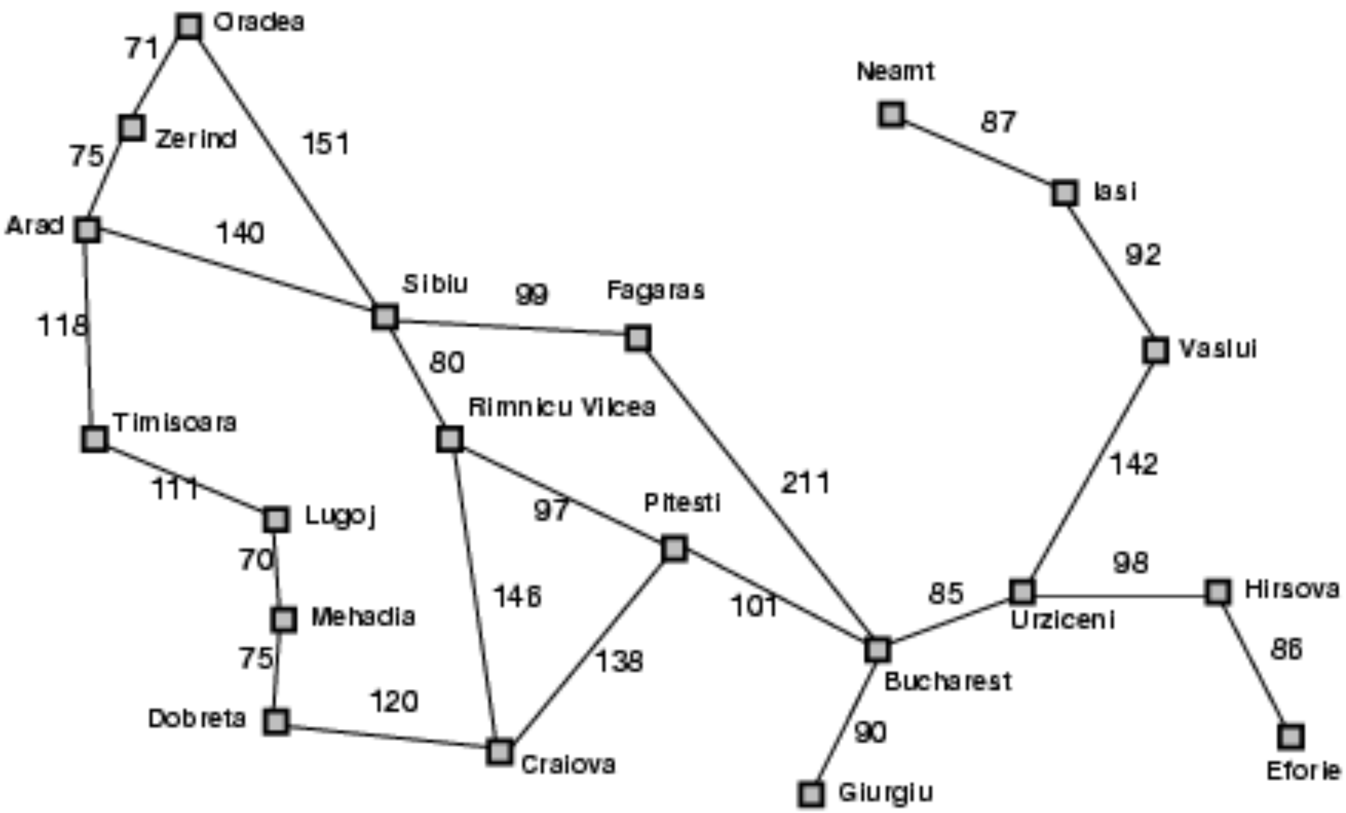
Goal



State space generated in heuristic search of the 8-puzzle graph.

Intelligent order of Expansion?

- TreeSearch (and GraphSearch) expands its nodes in a given order



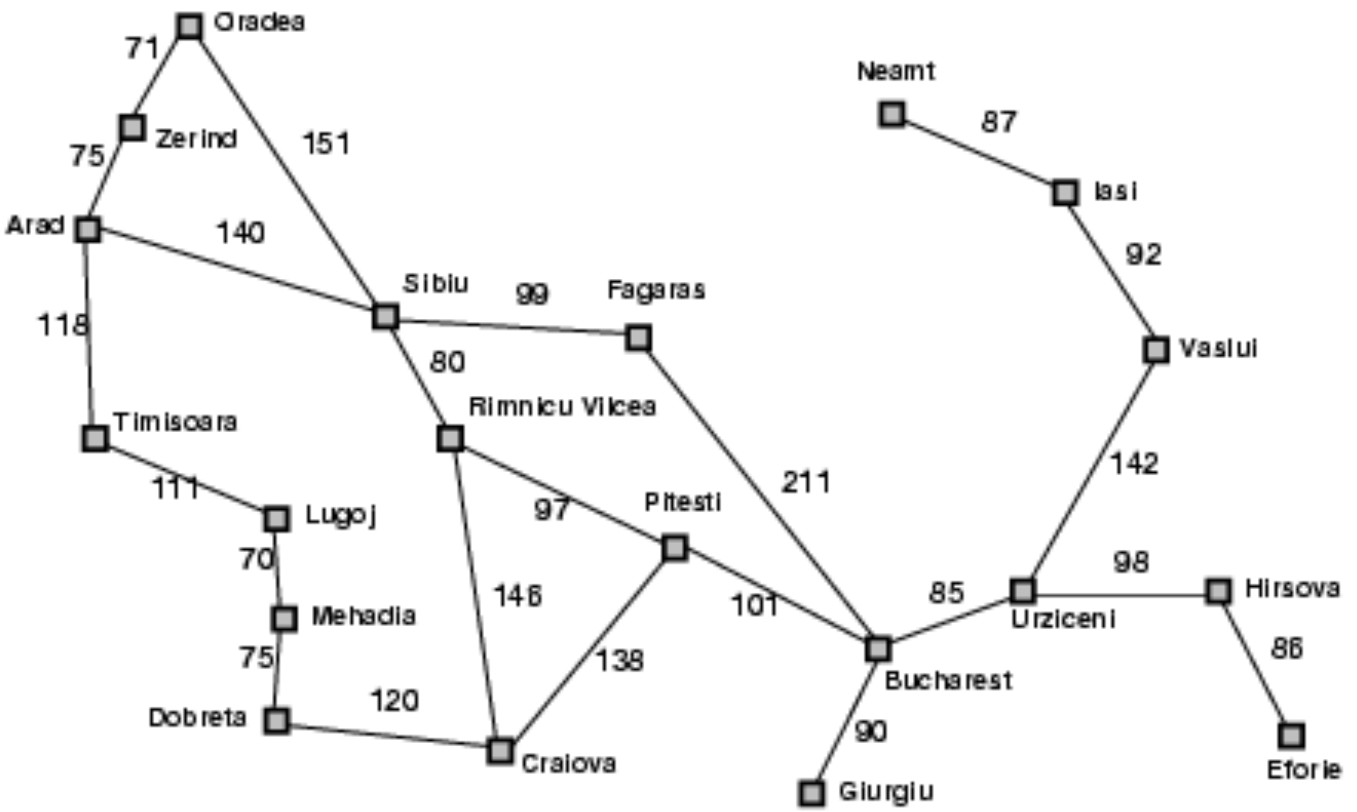
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Can we be "smart" about the order in which we expand nodes to improve the search?

tree search algorithm –keeping track of visited:

1. start w the initial node as curr
2. have I been to curr before? (is it in CLOSED)
3. is curr the goal?
4. if neither, expand curr - add children/ successors to OPEN, add curr to CLOSED
5. choose a node curr according to the smallest f(n) & go to step 2

Romania with step costs in km



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

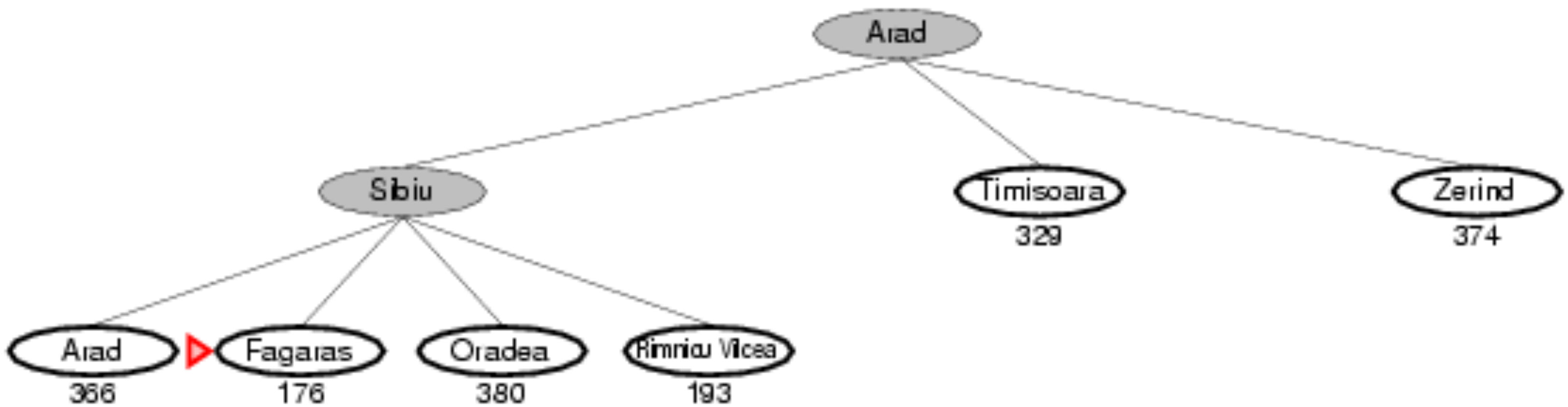
- Evaluation function $f(n) = h(n)$ (heuristic)

  = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

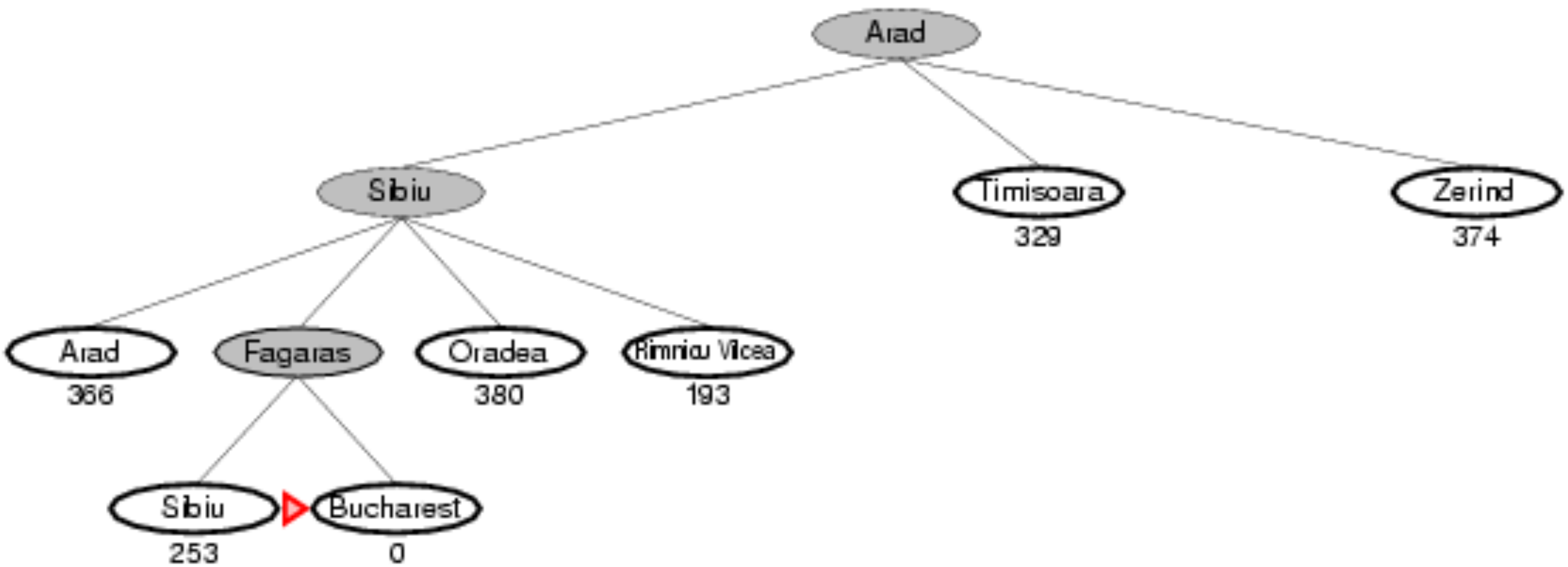
Greedy best-first search example

Greedy best-first search example

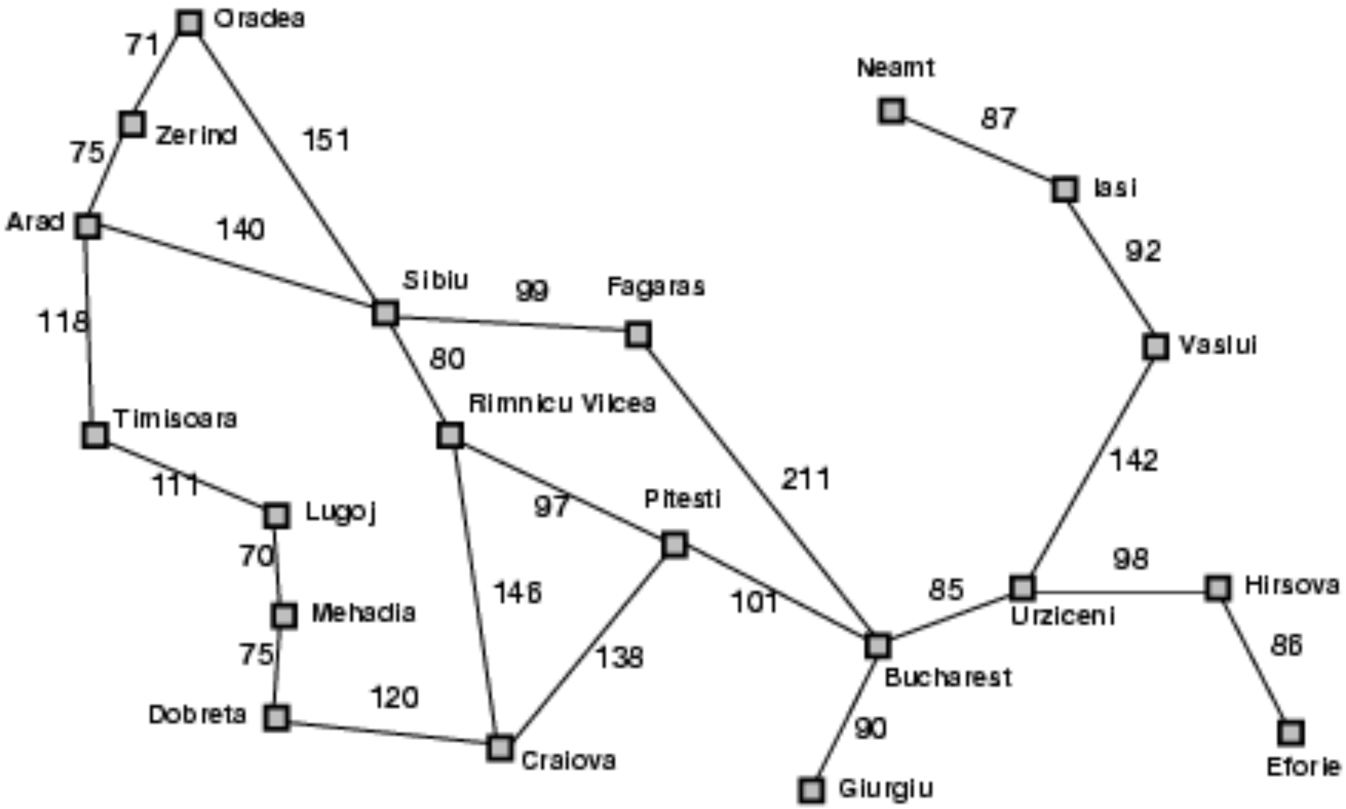
Greedy best-first search example



Greedy best-first search example



Problems with greedy best-first search?



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

SLD to Fagaras
Neamt – 180
Iasi – 200
Vasliu – 220
Fagaras - 0

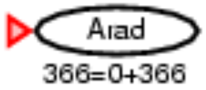
A^* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal (the evaluation of the desirability of n)

tree search algorithm –keeping track of visited:

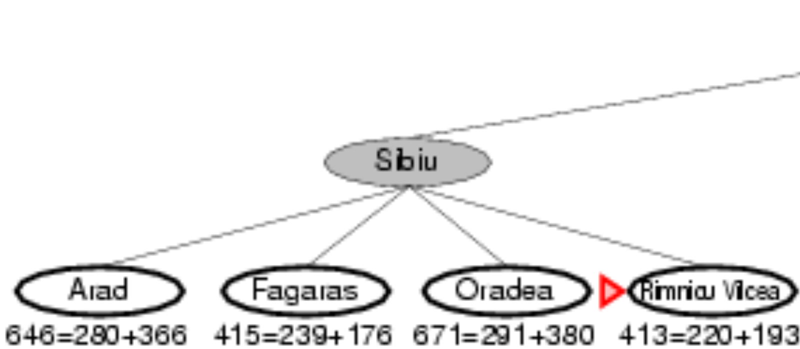
1. start w the initial node as curr
2. have I been to curr before? (is it in CLOSED)
3. is curr the goal?
4. if neither, expand curr - add children/ successors to OPEN, add curr to CLOSED
5. choose a node curr according to the smallest f(n) & go to step 2

A* search example

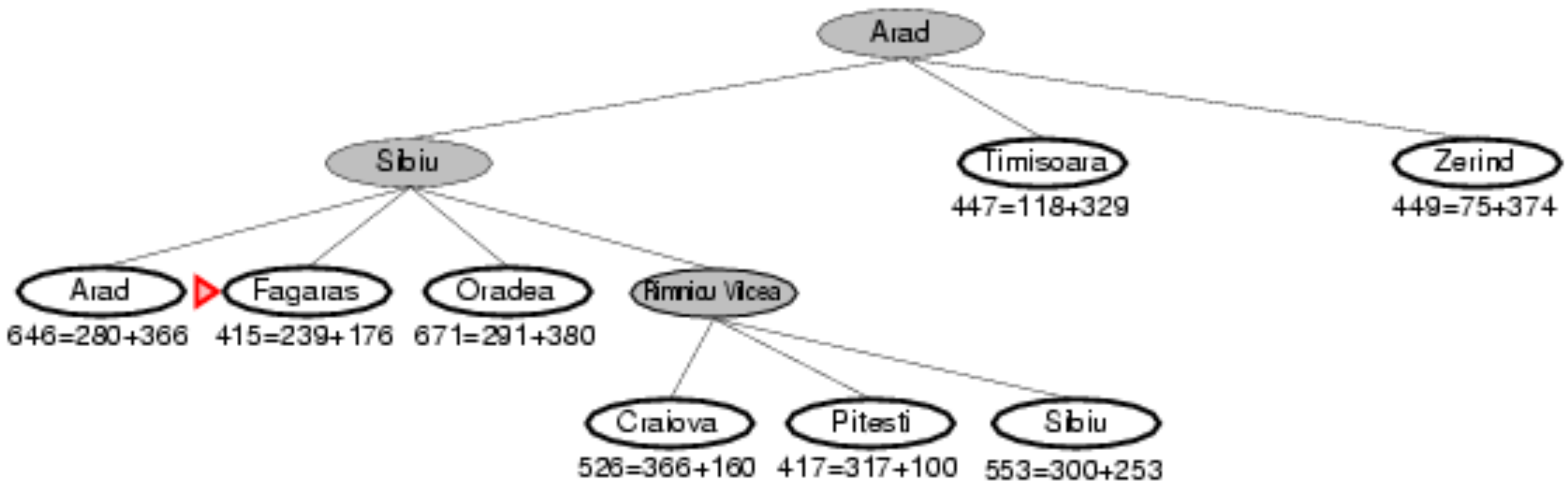


A* search example

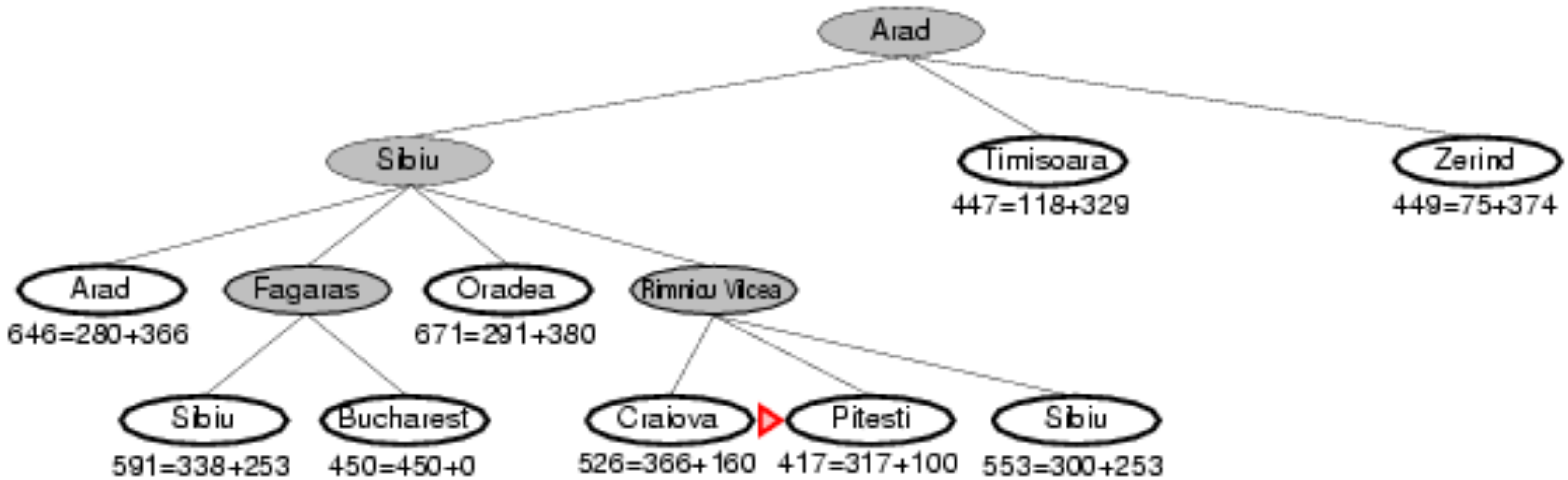
A* search example



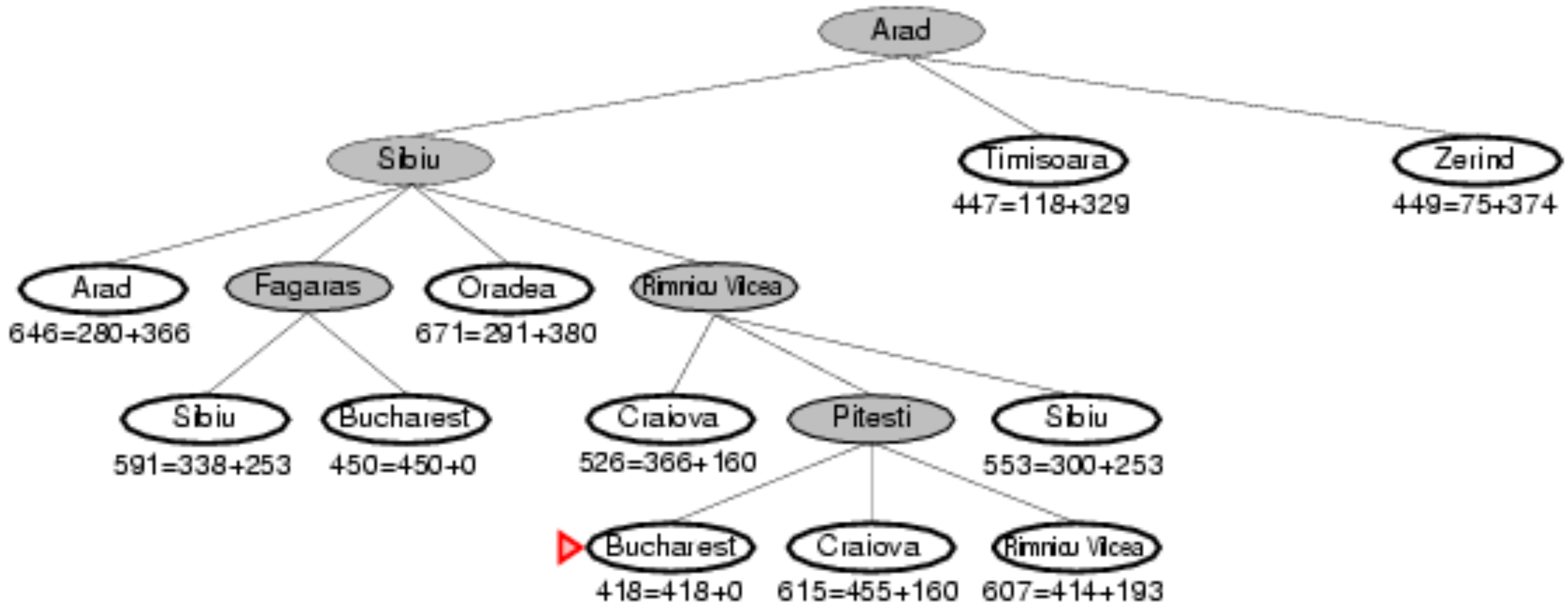
A* search example



A* search example



A* search example



Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n,

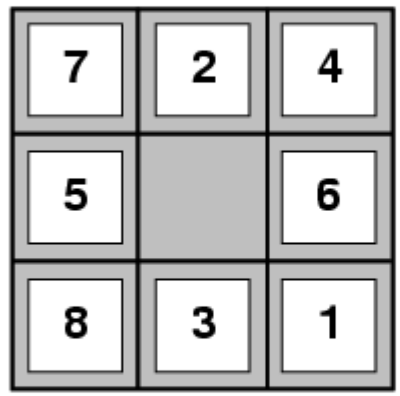
  $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

Admissible heuristics

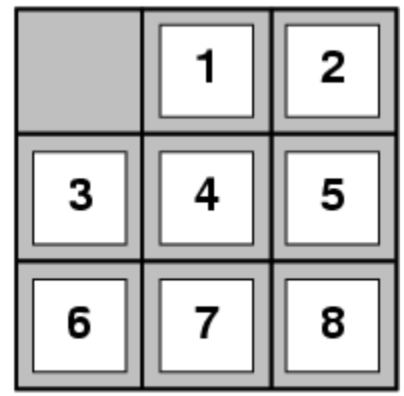
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

- <u>$h_1(S)$ = ?</u>
- <u>$h_2(S)$ = ?</u>

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search

Using A* in Planning