Informed search algorithms & Heuristic Functions

Chapter 3
3.5 and 3.6

Outline

- Best-first search
- Greedy best-first search
- A^* search
- Memory bounded Heuristics search
- Learning to search betters

Best first search

- Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function**, f(n).
- The evaluation function is interpreted as a cost estimate, so the node with the *lowest* evaluation is expanded first.
- Consider Heuristic function to be arbitrary, nonnegative, problem-specific functions, with one constraint: if n is a goal node, then h(n)=0.
- Most best-first algorithms include as a component of f a **heuristic function**, denoted h(n):
  - h(n) = estimated cost of the cheapest path from the state at node *n* to a goal state.

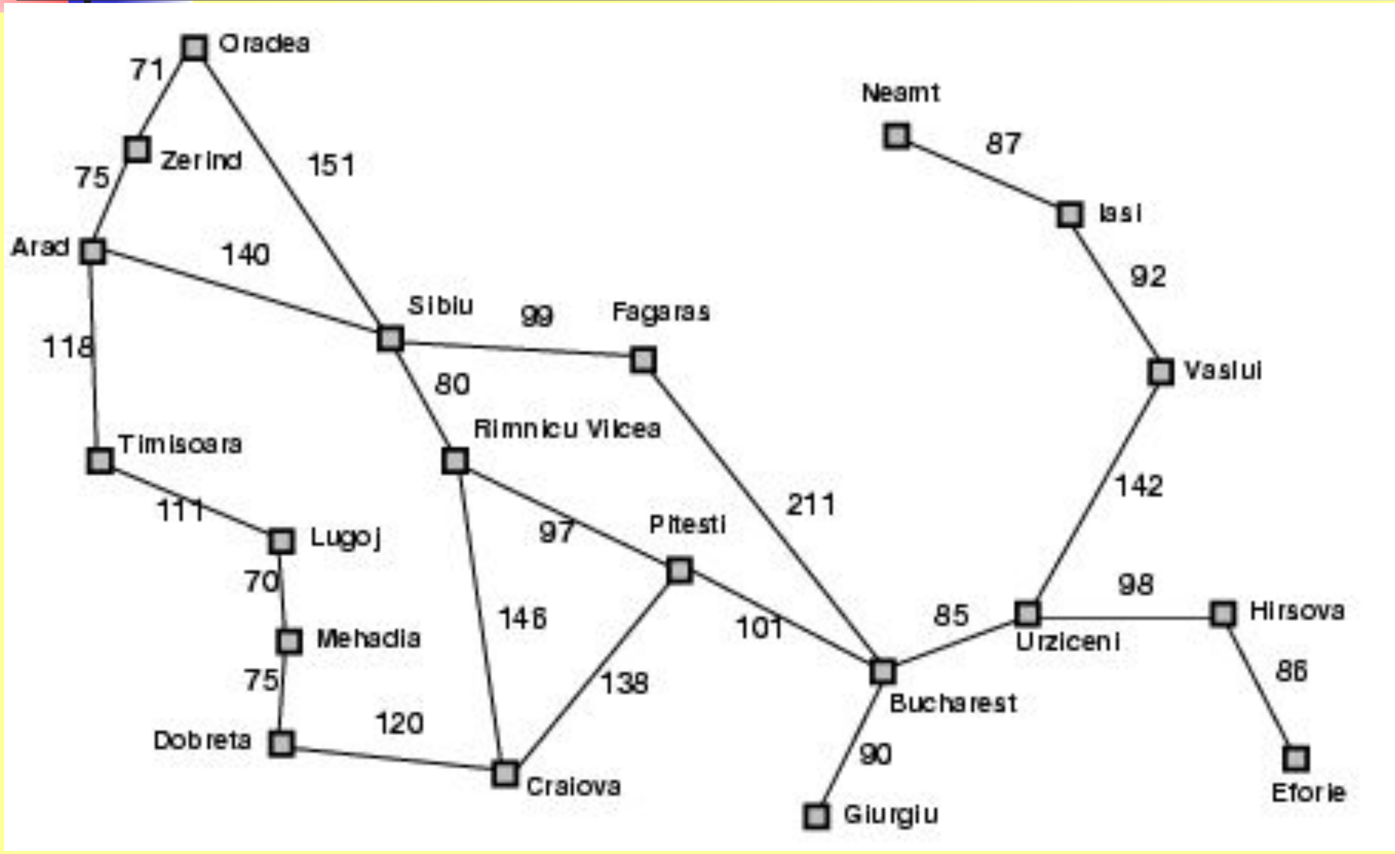
Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - f(n) provides an estimate for the total cost.
  - Expand the node n with smallest f(n).

- Implementation:
  Order the nodes in fringe increasing order of cost.

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

Romania with straight-line distance



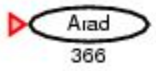
Straight–line distance
to Bucharest

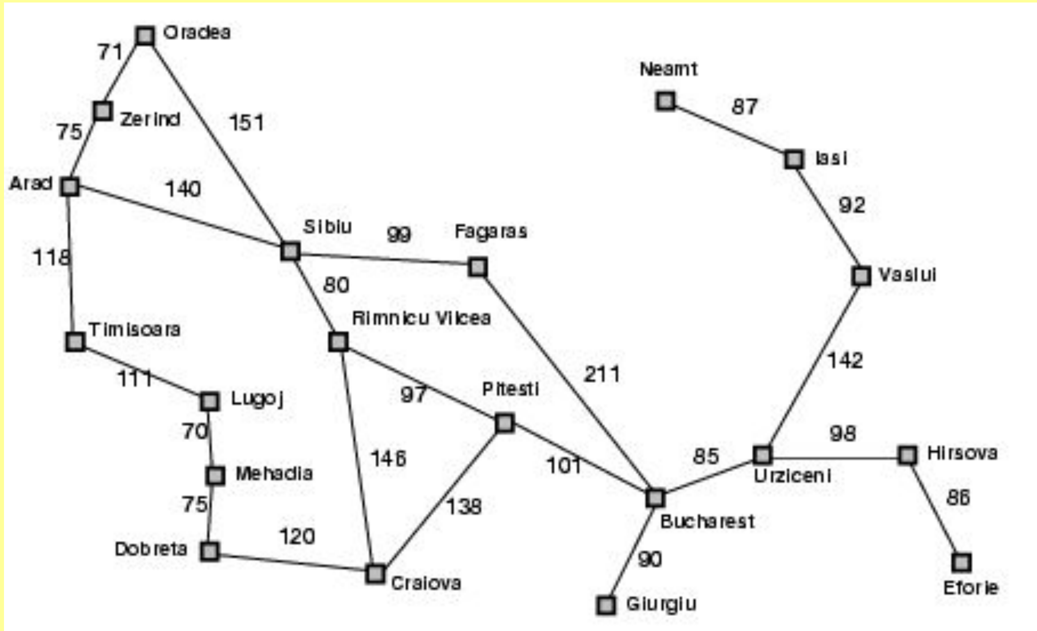
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- f(n) = estimate of cost from *n* to *goal*
- e.g., $f_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal.

Greedy best-first search example

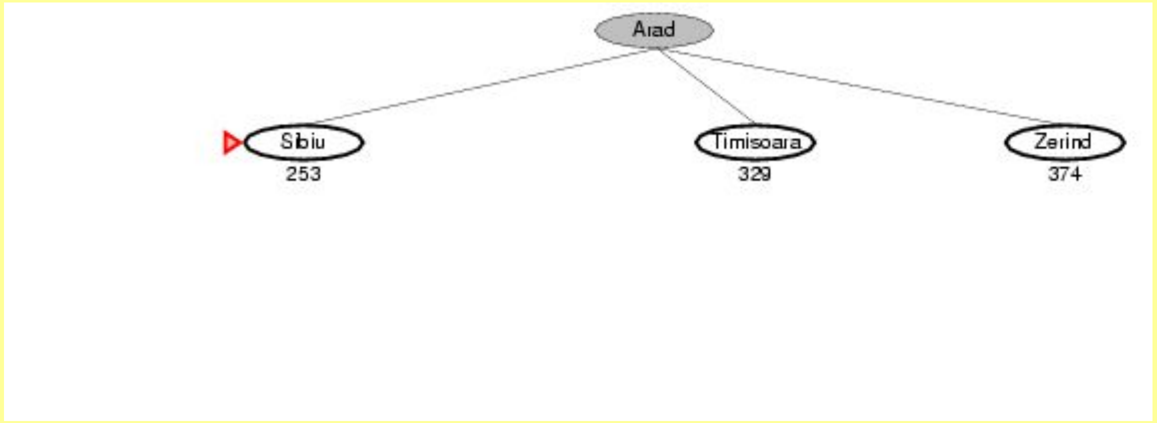


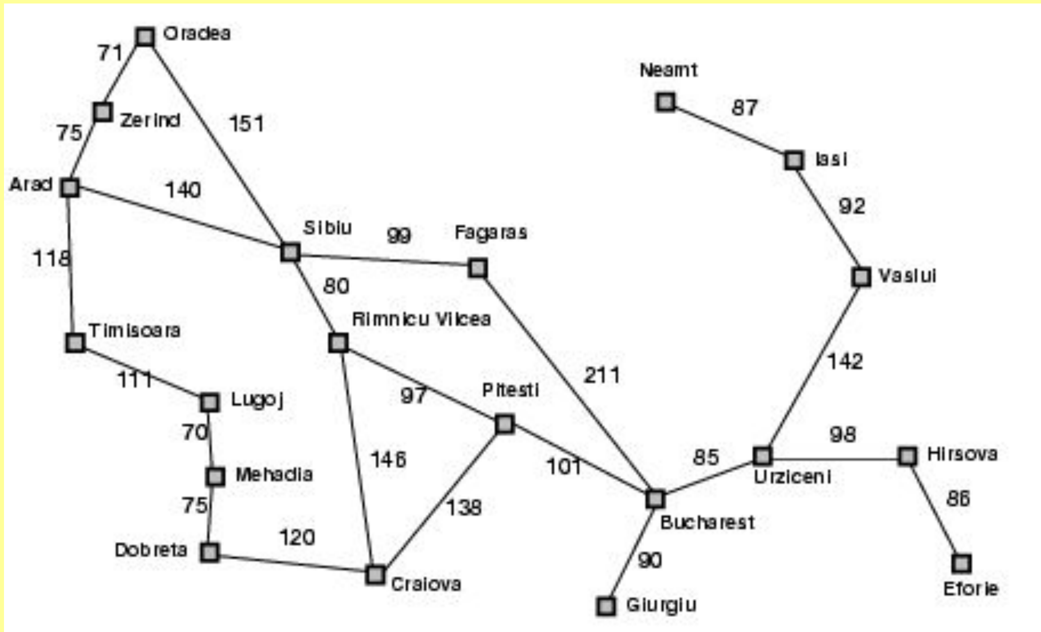


Straight–line distance to Bucharest

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 example

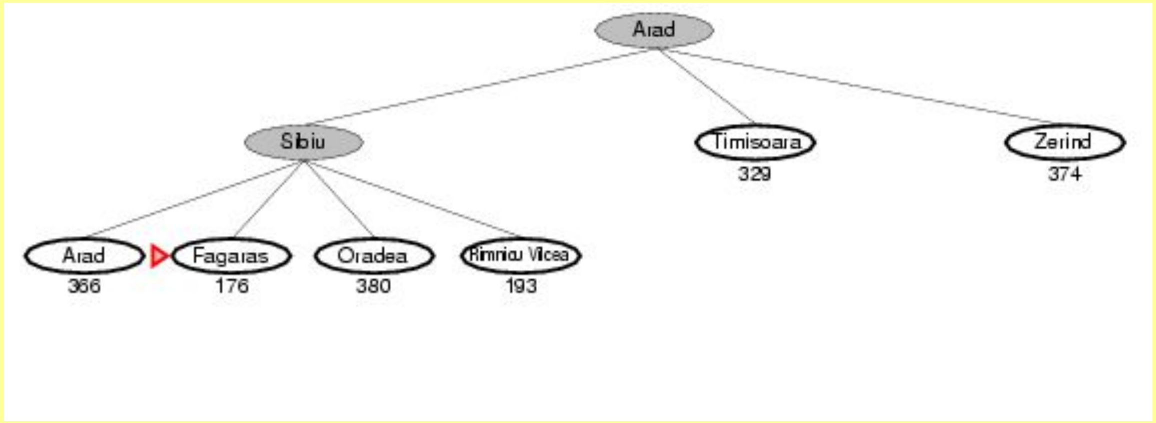


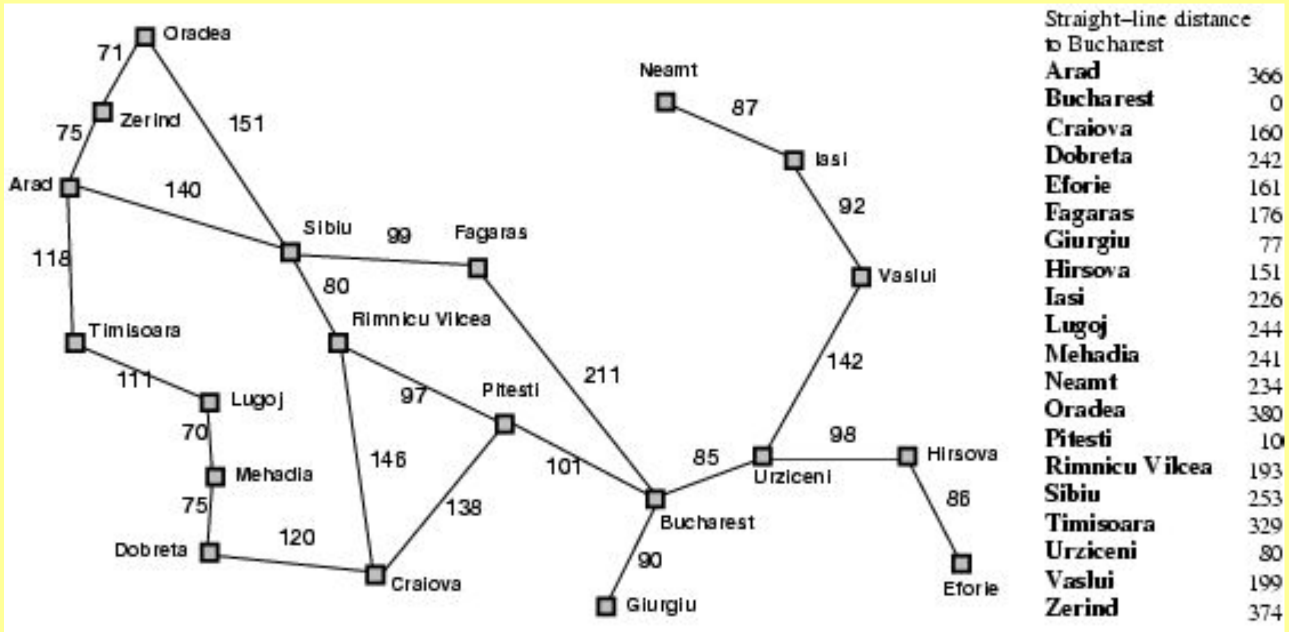


Straight–line distance to Bucharest

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 example

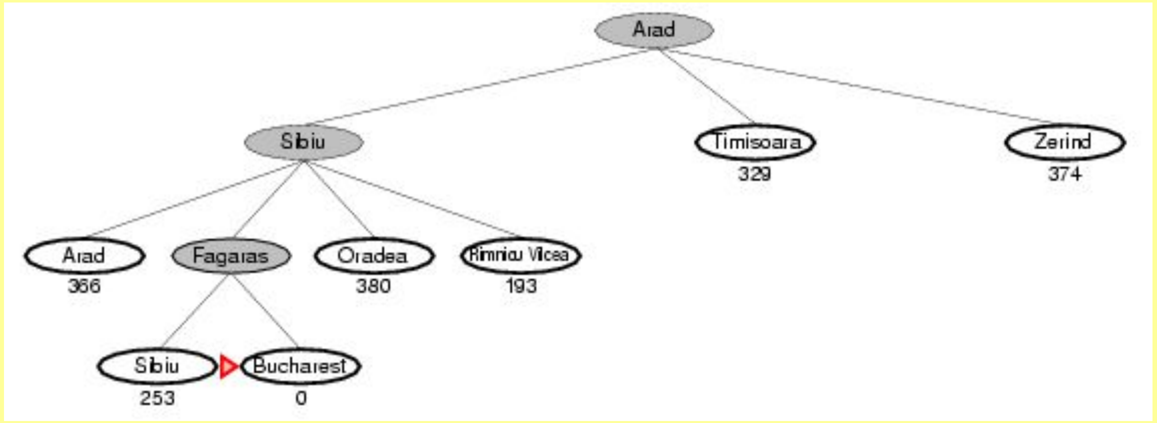


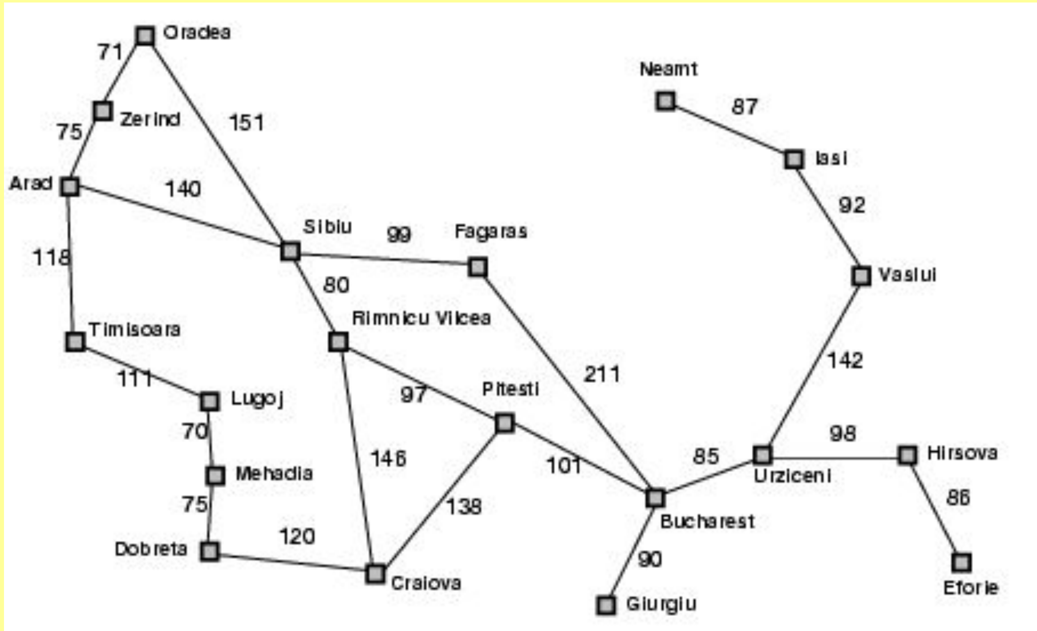


Straight–line distance to Bucharest

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 example





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

Properties of greedy best-first search

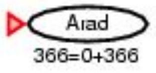
- Complete? No – can get stuck in loops / dead end. (Iasi to Fagarus)
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ - keeps all nodes in memory
- Optimal? No

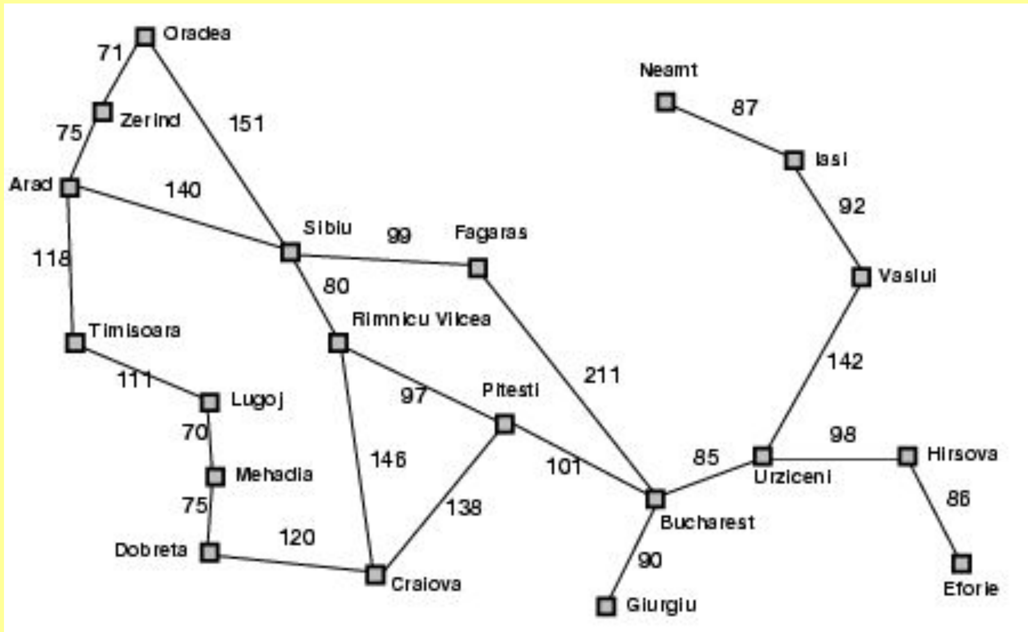
e.g. Arad☐Sibiu☐Rimnicu Virea☐Pitesti☐Bucharest is shorter!

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
- Best First search has $f(n)=h(n)$

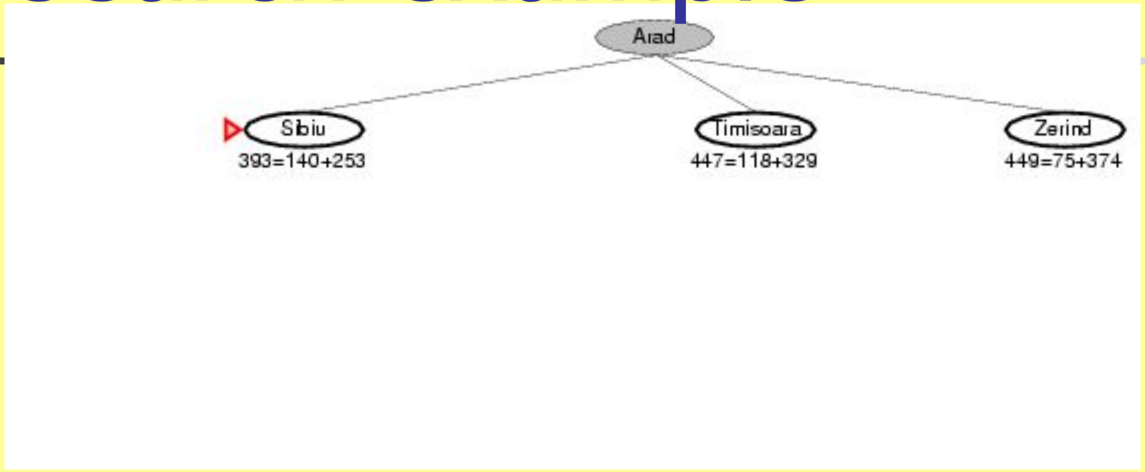
A* search example

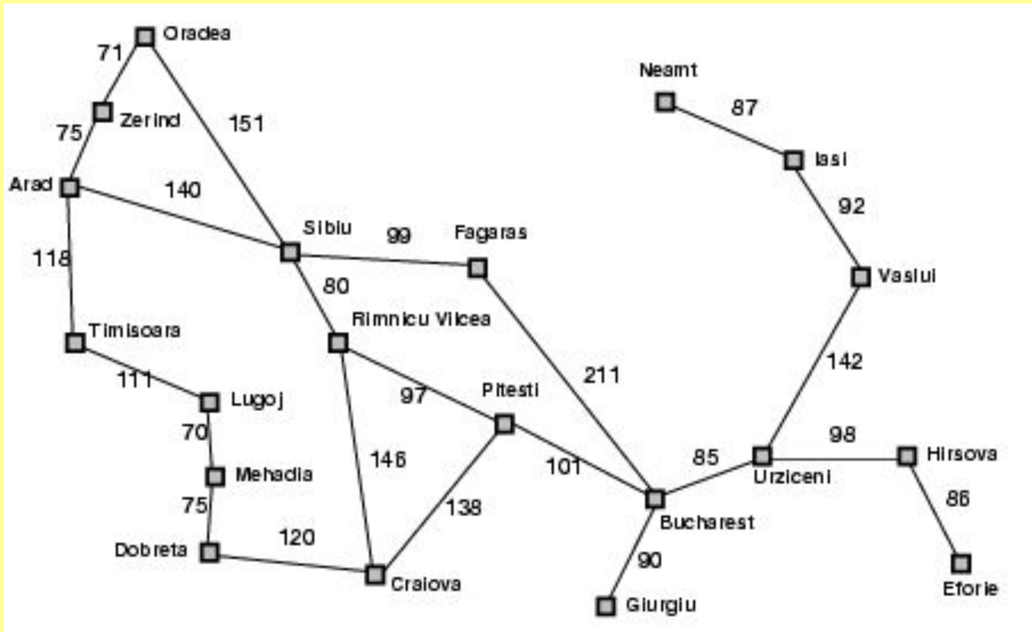




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

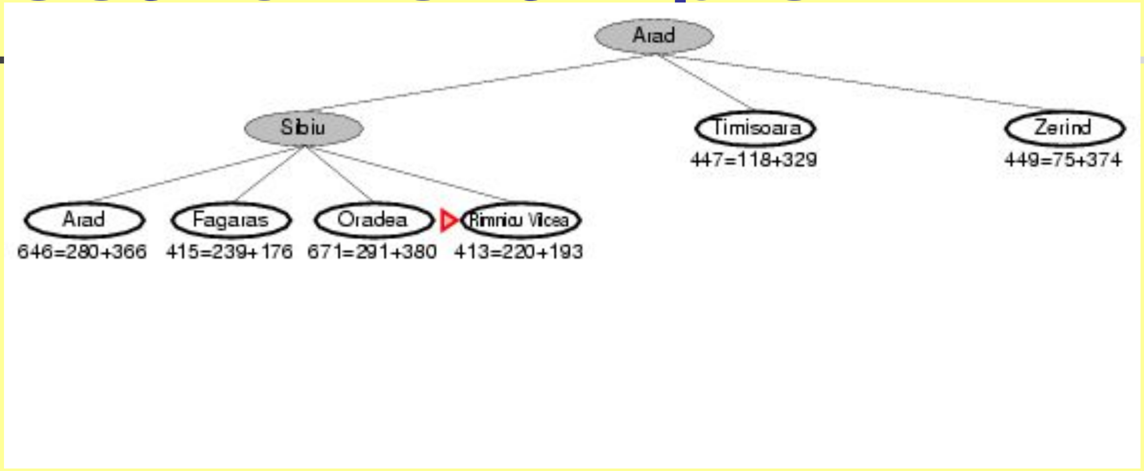


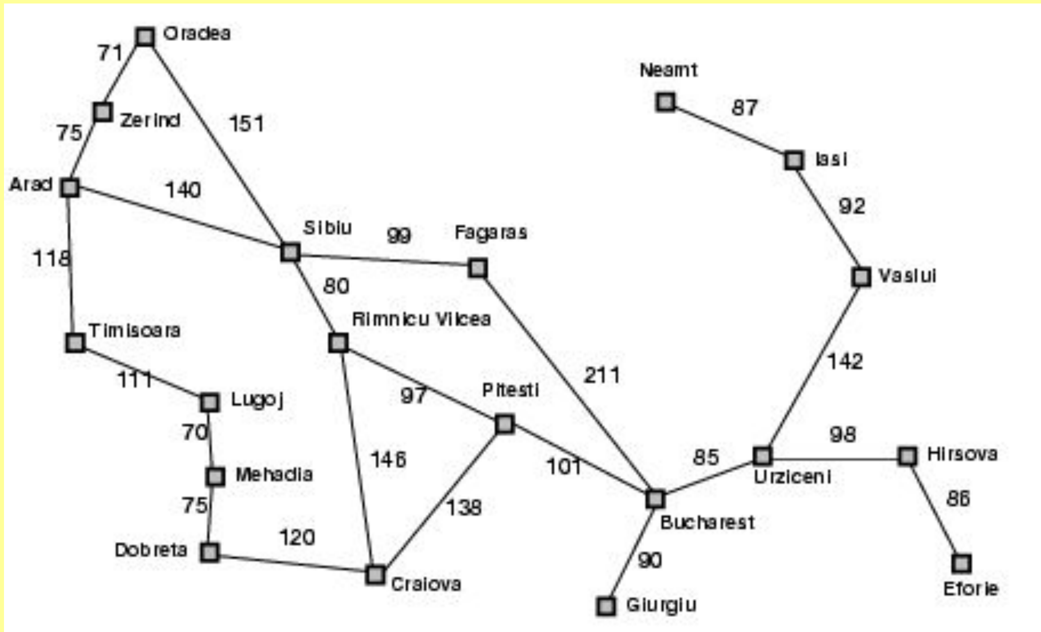


Straight−line distance
to Bucharest

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

A* search example

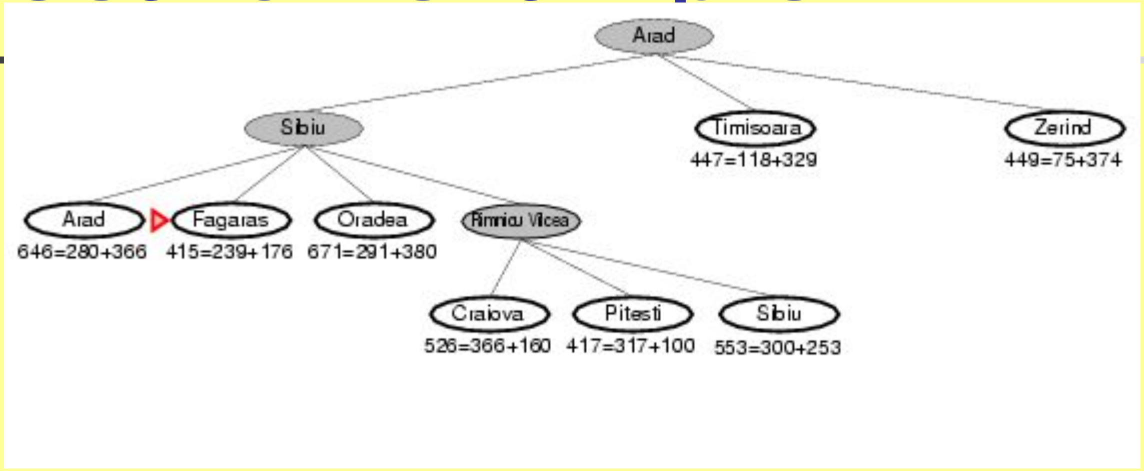


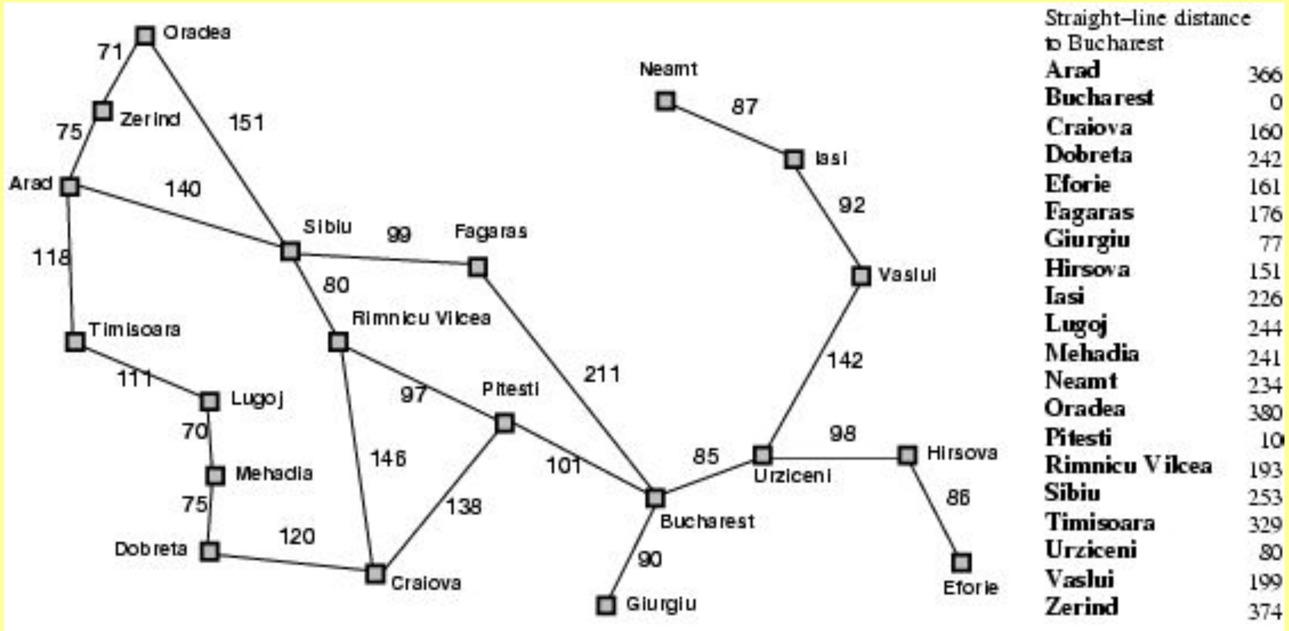


Straight–line distance to Bucharest

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

A* search example

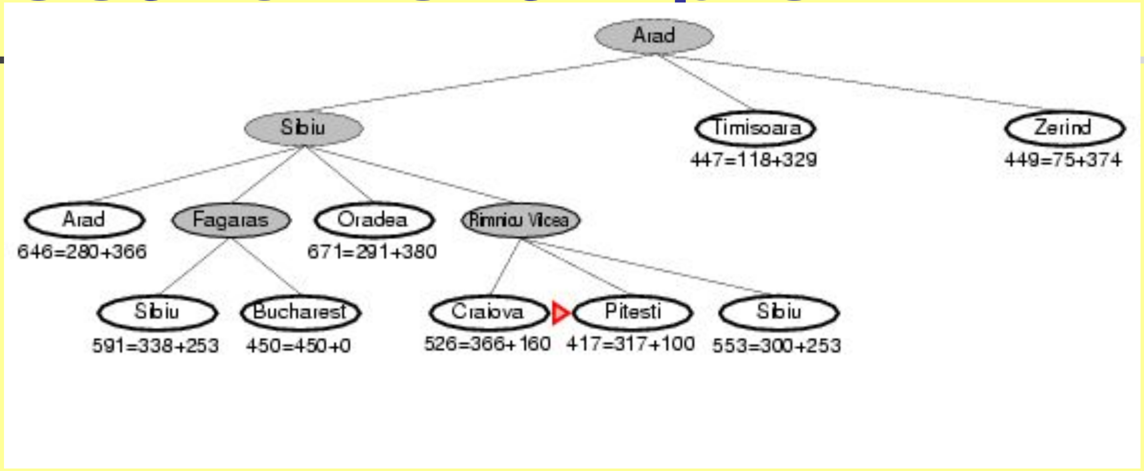


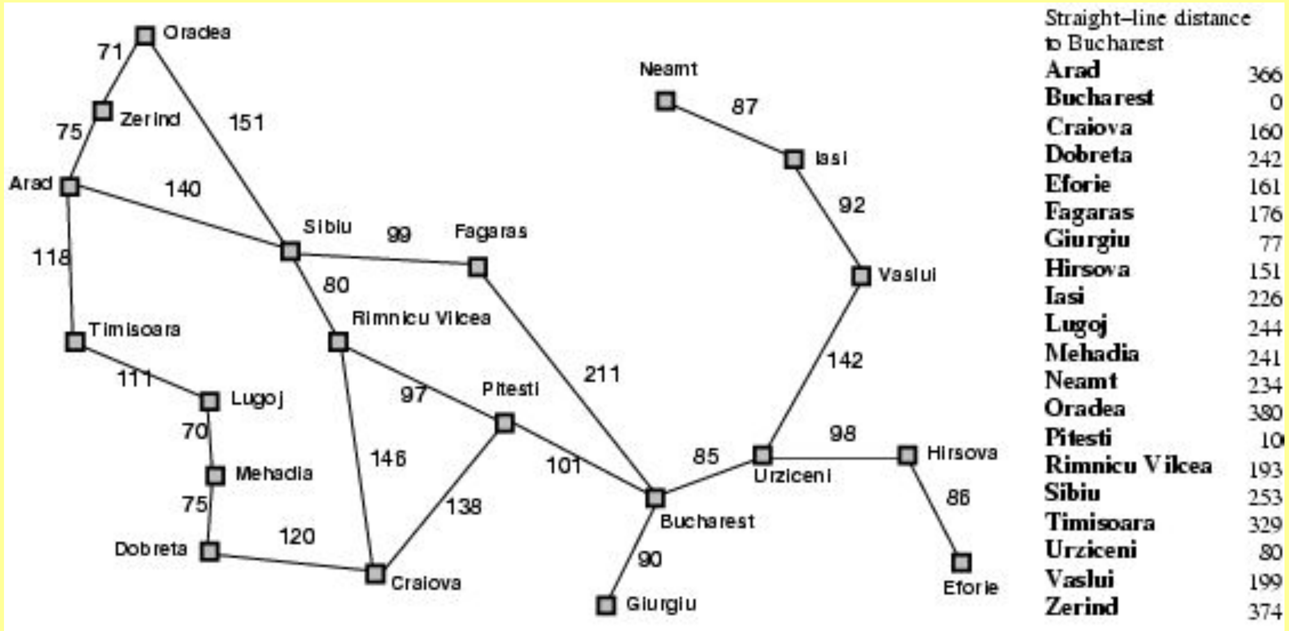


Straight–line distance to Bucharest

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	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

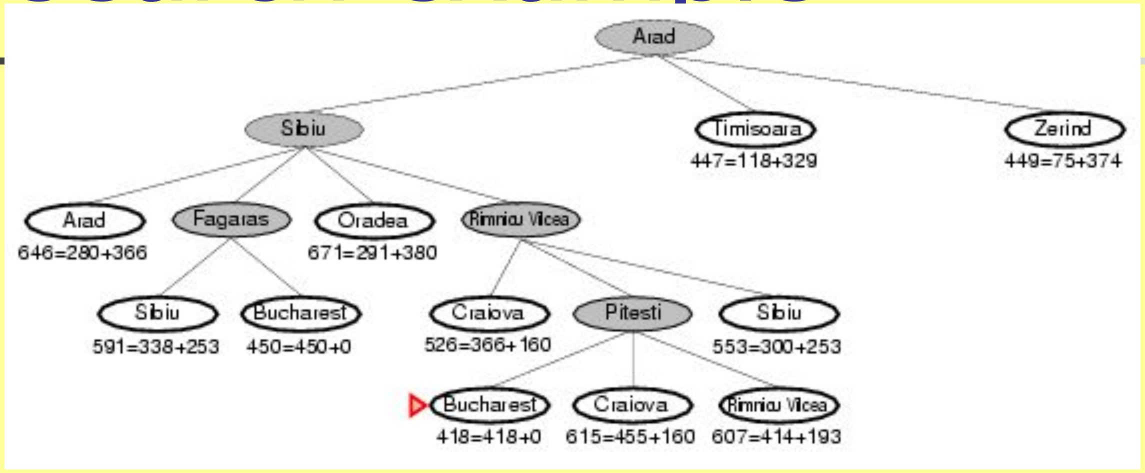
A* search example

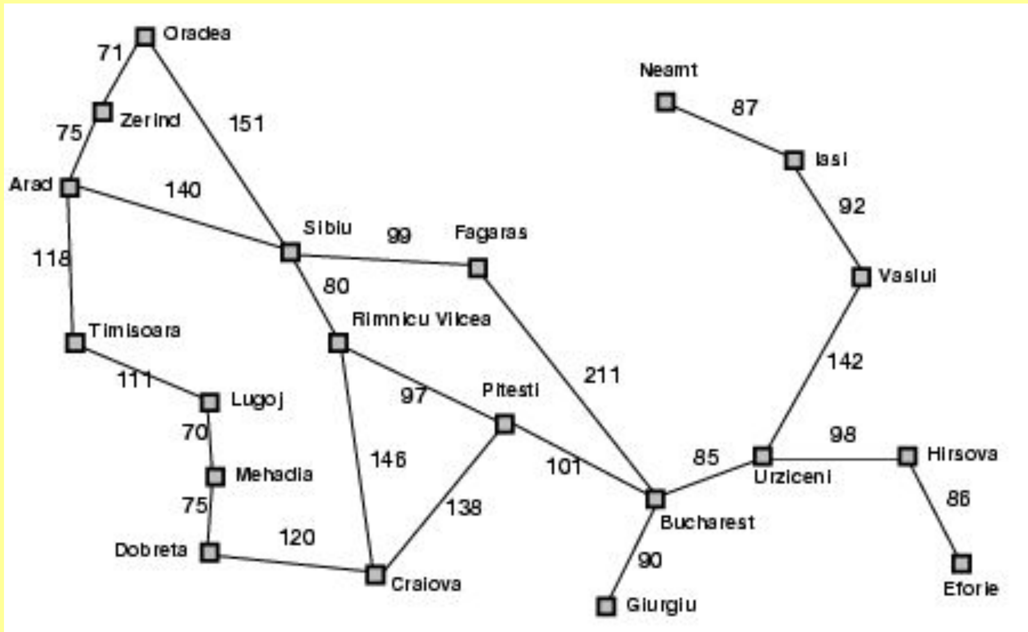




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example





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

End of class

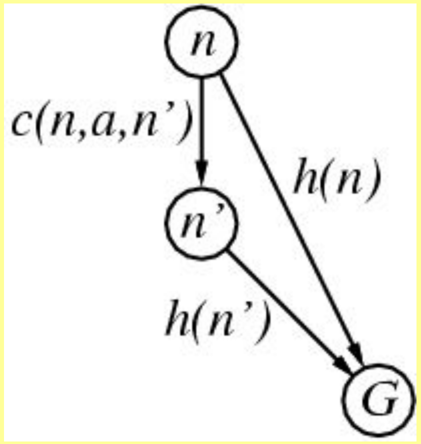
Admissible heuristics

- 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)
- Theorem: If $h(n)$ is admissible, A* using `TREE-SEARCH` is optimal *while the graph-search version is optimal if* h(n) *is consistent.*

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

  $h(n) \le c(n,a,n') + h(n')$



It's the triangle inequality !

- If *h* is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) = f(n) \\ f(n') &\ge f(n) \end{aligned}$$

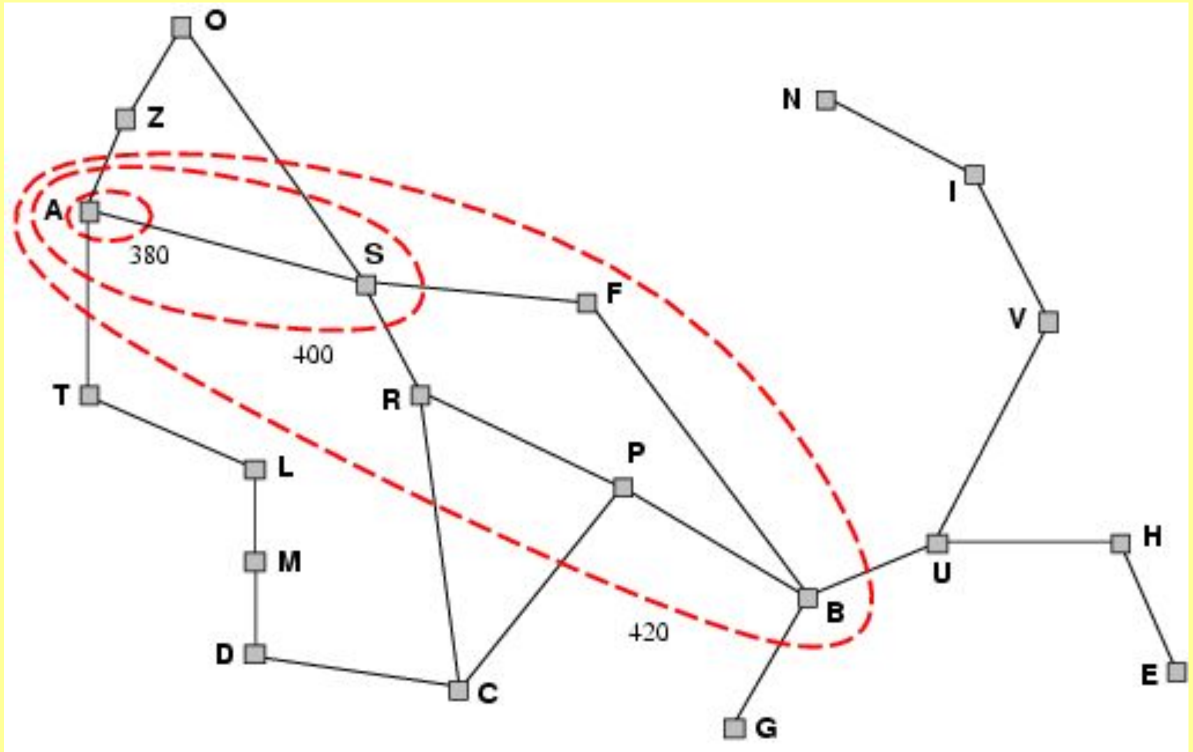
- i.e., *f(n)* is non-decreasing along any path.

- Theorem:
  If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal

keeps all checked nodes in memory to avoid repeated states

Optimality of A*

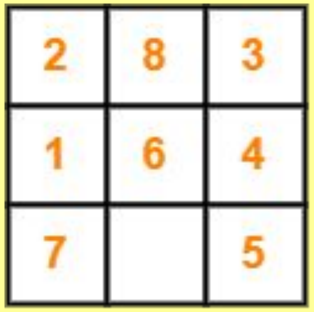
- A* expands nodes in order of increasing *f* value
- Gradually adds "*f*-contours" of nodes
- Contour *i* contains all nodes with $f \le f_i$ where $f_i < f_{i+1}$



Problem of A*

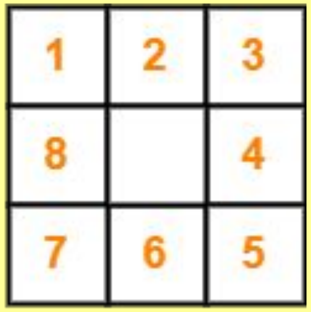
reach the final state from initial state using A* Algorithm.

Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.



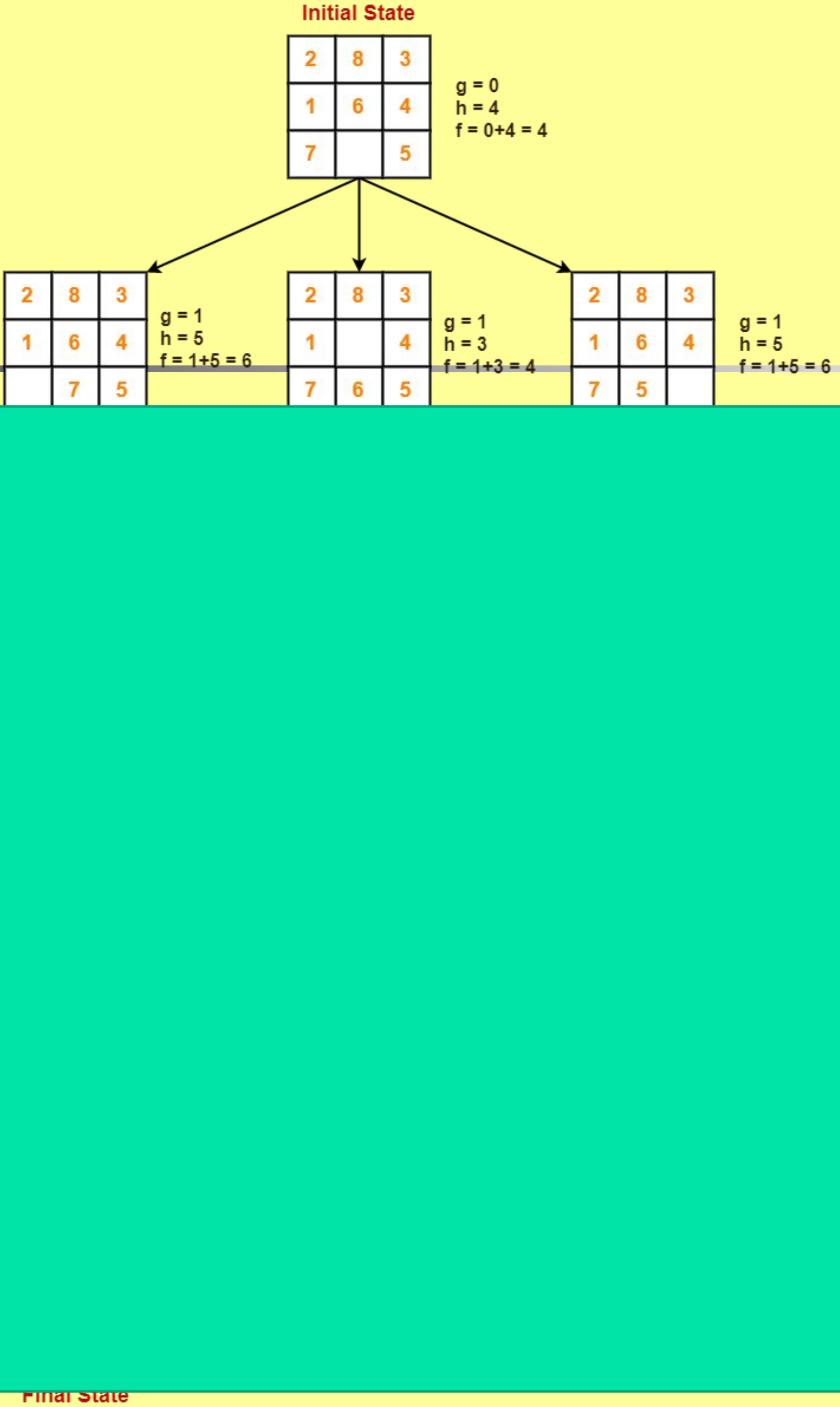
2	8	3
1	6	4
7		5

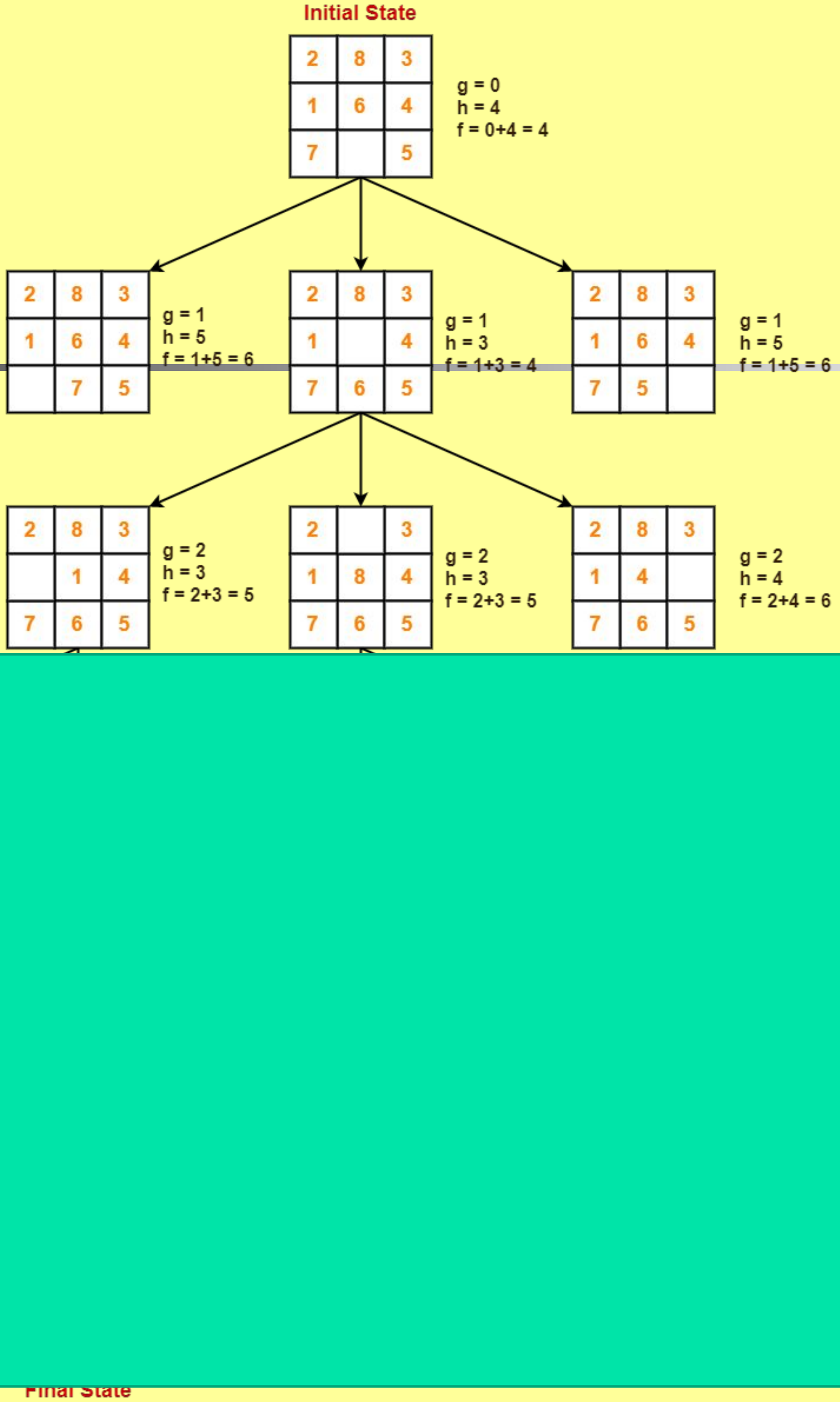
Initial State



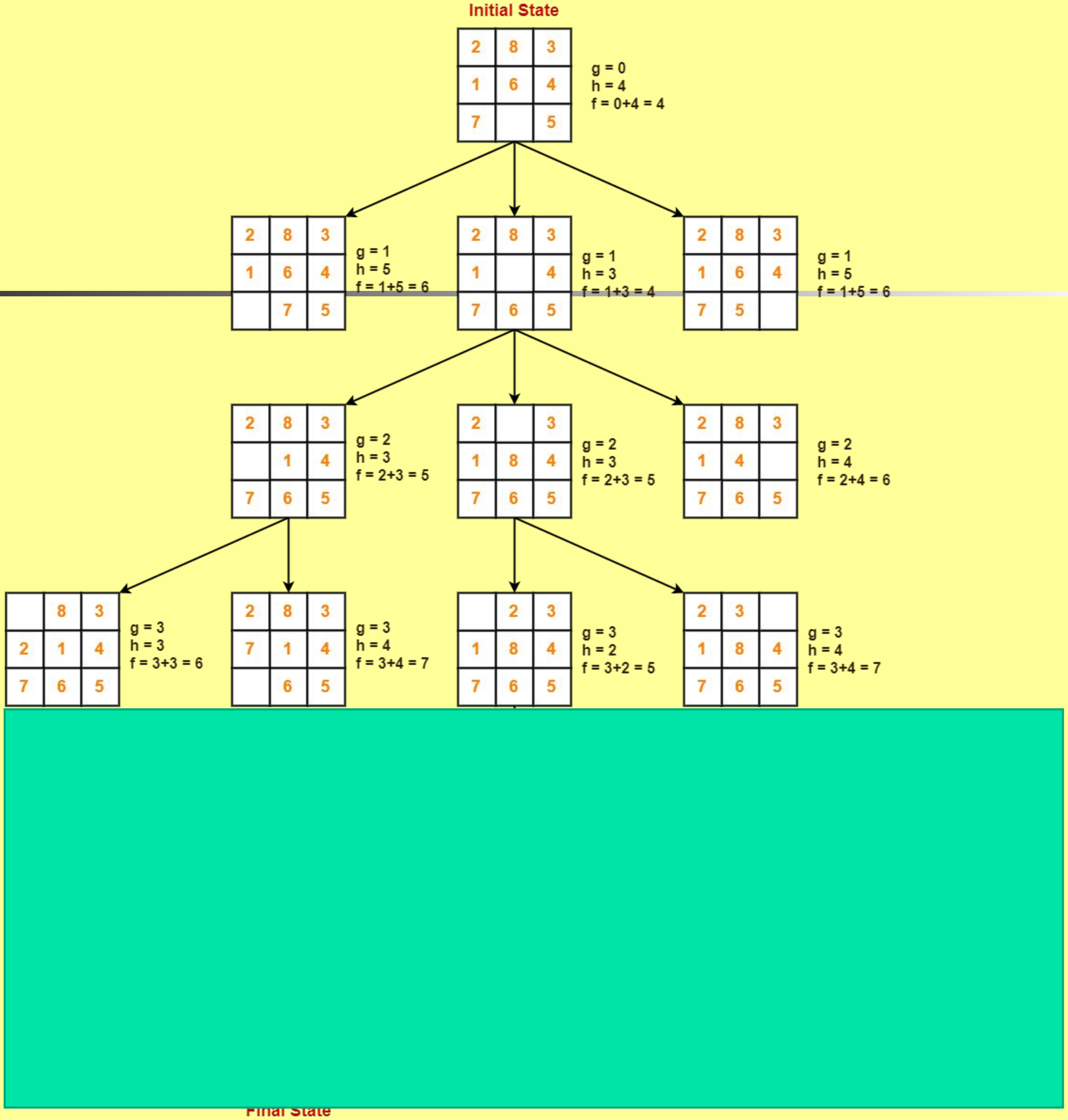
1	2	3
8		4
7	6	5

Final State

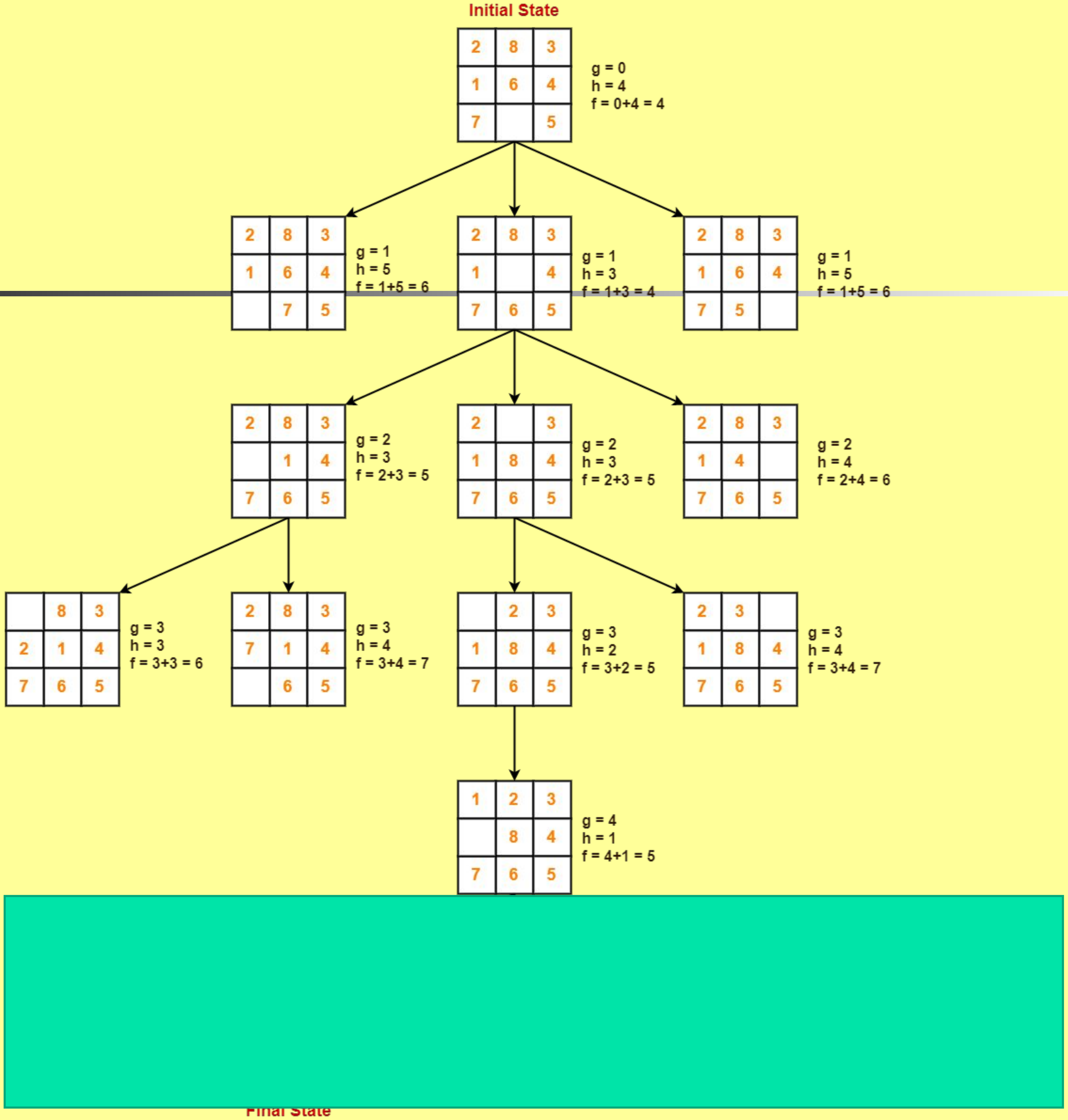

Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
Final State

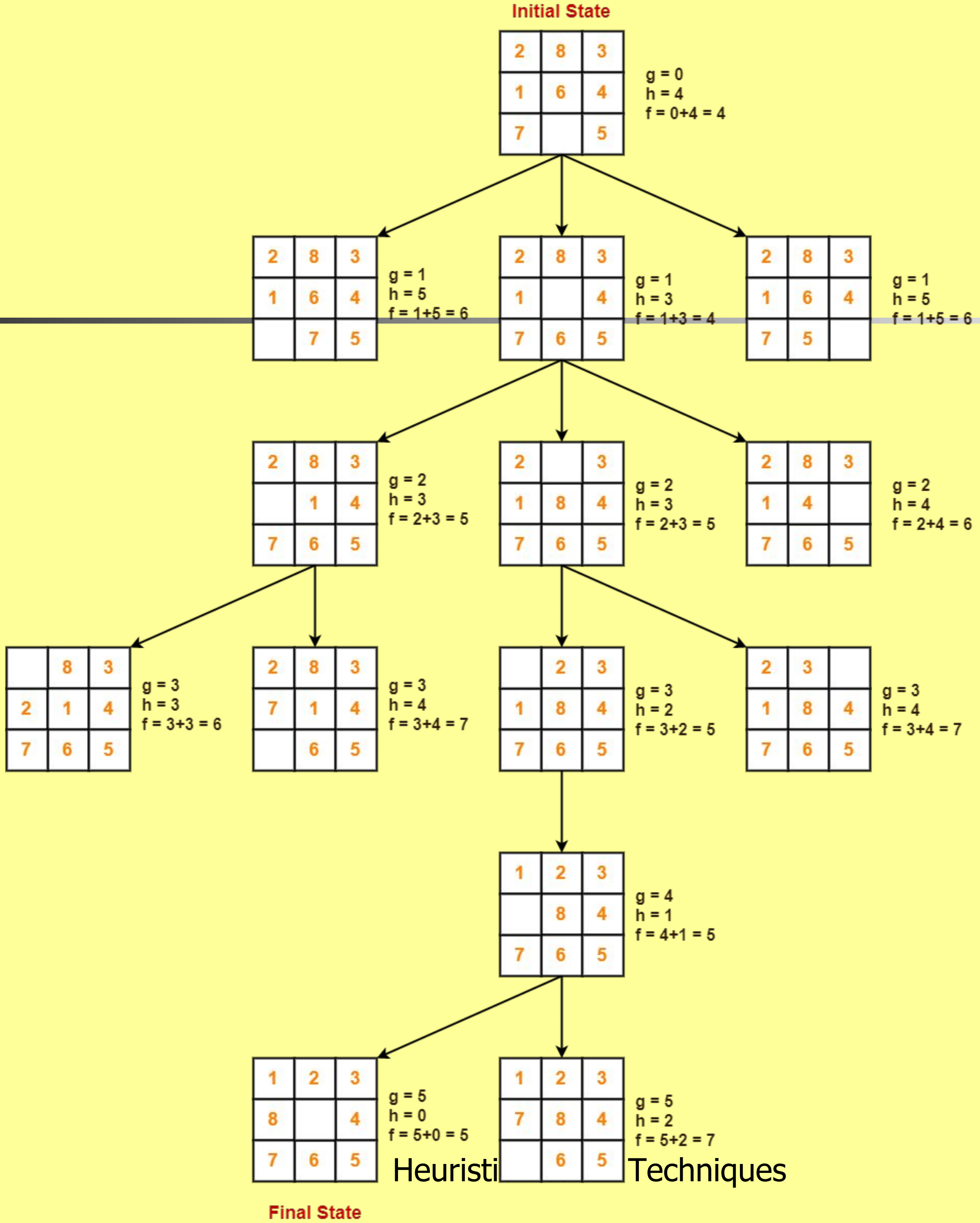


Initial State
g = 0
h = 4
f = 0+4 = 4
g = 1
h = 5
f = 1+5 = 6
g = 1
h = 3
f = 1+3 = 4
g = 1
h = 5
f = 1+5 = 6
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 4
f = 2+4 = 6
Final State



Initial State
g = 0
h = 4
f = 0+4 = 4
g = 1
h = 5
f = 1+5 = 6
g = 1
h = 3
f = 1+3 = 4
g = 1
h = 5
f = 1+5 = 6
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 4
f = 2+4 = 6
g = 3
h = 3
f = 3+3 = 6
g = 3
h = 4
f = 3+4 = 7
g = 3
h = 2
f = 3+2 = 5
g = 3
h = 4
f = 3+4 = 7
Final State


Initial State
2 8 3 / 1 6 4 / 7 _ 5
g = 0
h = 4
f = 0+4 = 4
2 8 3 / 1 6 4 / _ 7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3 / 1 _ 4 / 7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3 / 1 6 4 / 7 5 _
g = 1
h = 5
f = 1+5 = 6
2 8 3 / _ 1 4 / 7 6 5
g = 2
h = 3
f = 2+3 = 5
2 _ 3 / 1 8 4 / 7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3 / 1 4 _ / 7 6 5
g = 2
h = 4
f = 2+4 = 6
_ 8 3 / 2 1 4 / 7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3 / 7 1 4 / _ 6 5
g = 3
h = 4
f = 3+4 = 7
_ 2 3 / 1 8 4 / 7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3 _ / 1 8 4 / 7 6 5
g = 3
h = 4
f = 3+4 = 7
1 2 3 / _ 8 4 / 7 6 5
g = 4
h = 1
f = 4+1 = 5
Final State


Initial State
g = 0
h = 4
f = 0+4 = 4
g = 1
h = 5
f = 1+5 = 6
g = 1
h = 3
f = 1+3 = 4
g = 1
h = 5
f = 1+5 = 6
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 3
f = 2+3 = 5
g = 2
h = 4
f = 2+4 = 6
g = 3
h = 3
f = 3+3 = 6
g = 3
h = 4
f = 3+4 = 7
g = 3
h = 2
f = 3+2 = 5
g = 3
h = 4
f = 3+4 = 7
g = 4
h = 1
f = 4+1 = 5
g = 5
h = 0
f = 5+0 = 5
g = 5
h = 2
f = 5+2 = 7
Final State

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f $\leq$ *f(G)* , i.e. path-cost > ε)
- Time/Space? Exponential b^d
- Optimal? Yes
- *Optimally Efficient*: Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)

Memory Bounded Heuristic Search: Recursive BFS

- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.
- We remember the best f-value we have found so far in the branch we are deleting.

Types of Memory Bounded algorithms

- IDA* - Is practical for many problems with unit step costs and avoids the substantial overhead associated with keeping a sorted queue of nodes. Unfortunately, it suffers from the same difficulties with real valued costs as does the iterative version of uniform-cost search
- RBFS
- MA*

Difference between IDA* and IDDFS

- The main difference between IDA∗ and standard iterative deepening is that the cutoff used is the f-cost (g+h) rather than the depth;
- At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

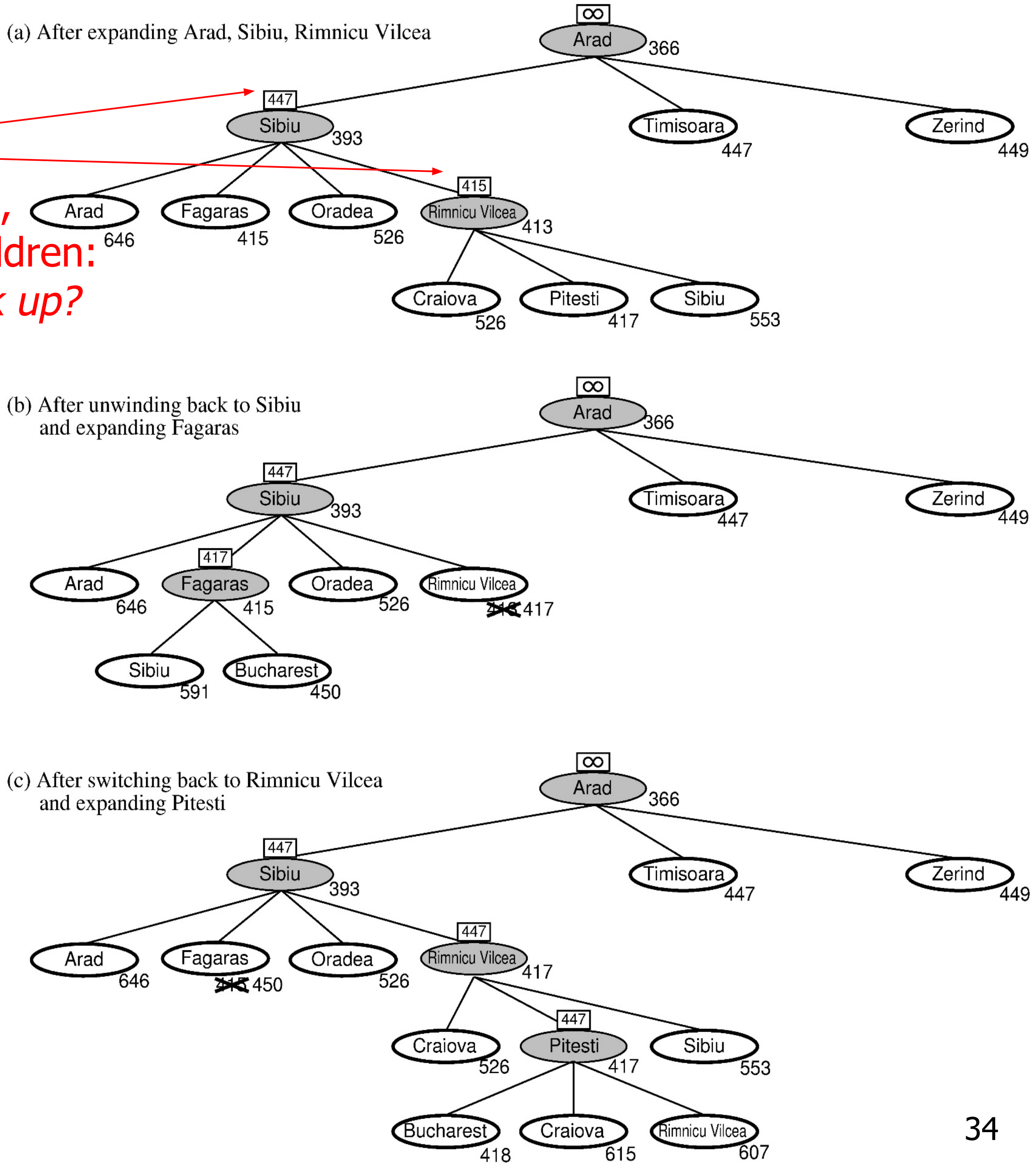
RBFS:

best alternative
over fringe nodes,
which are not children:
do I want to back up?

RBFS changes its mind very often in practice.

This is because the f=g+h become more accurate (less optimistic) as we approach the goal. Hence, higher level nodes have smaller f-values and will be explored first.

Problem: We should keep in memory whatever we can.



End of class

RBFS

- Suffers from excessive node regeneration.
- Like A^* tree search, RBFS is an optimal algorithm if the heuristic function h(n) is admissible.
- Its space complexity is linear in the depth of the deepest optimal solution
- Time complexity is rather difficult to characterize: it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded.

IDA* and RBFS- Drawback

- IDA* and RBFS suffer from using *too little* memory.
- To utilize available memory
  - **MA*** (memory-bounded A*)
  - **SMA*** (simplified MA*)

Properties of SMA*

- SMA* expands the best leaf and deletes the worst leaf.
- SMA* expands the *newest* best leaf and deletes the *oldest* worst leaf – if F-value same
- SMA* is complete if there is any reachable solution
- It is optimal if any optimal solution is reachable; otherwise, it returns the best reachable solution.
- Time and space complexity is inescapable problem if subset is regenerated.

Learning to search better

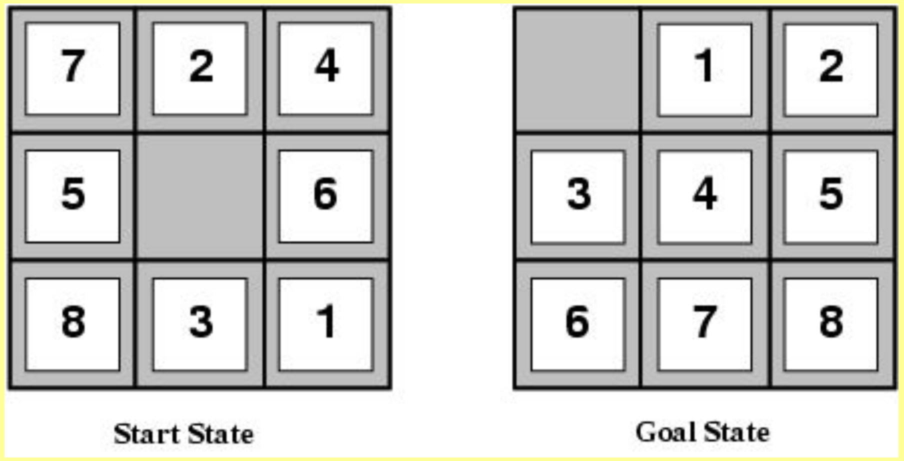
- Could an agent *learn* how to search better?
- **Metalevel state space** captures the internal (computational) state of a program that is searching in an **object-level state space** such as Romania.
- For example, the internal state of the A^* algorithm consists of the current search tree. Each action in the metalevel state space is a computation step that alters the internal state; for example, each computation step in A^* expands a leaf node and adds its successors to the tree.
- **Metalevel learning** algorithm- The goal of learning is to minimize the **total cost** of problem solving, trading off computational expense and path cost.

3.6 Heuristic Functions - Admissible heuristics

E.g., for the 8-puzzle:

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

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



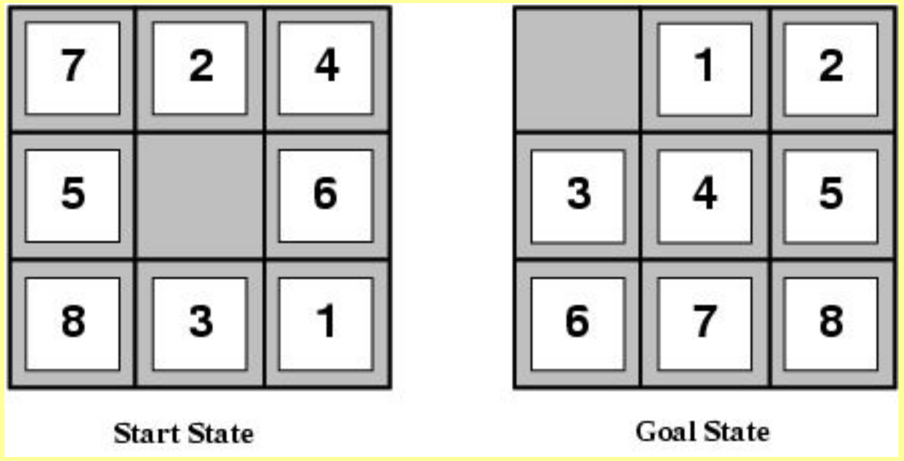
- $h_1(S) = ?$
- $h_2(S) = ?$

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)



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

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: it is guaranteed to expand less nodes.

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Relaxed problems

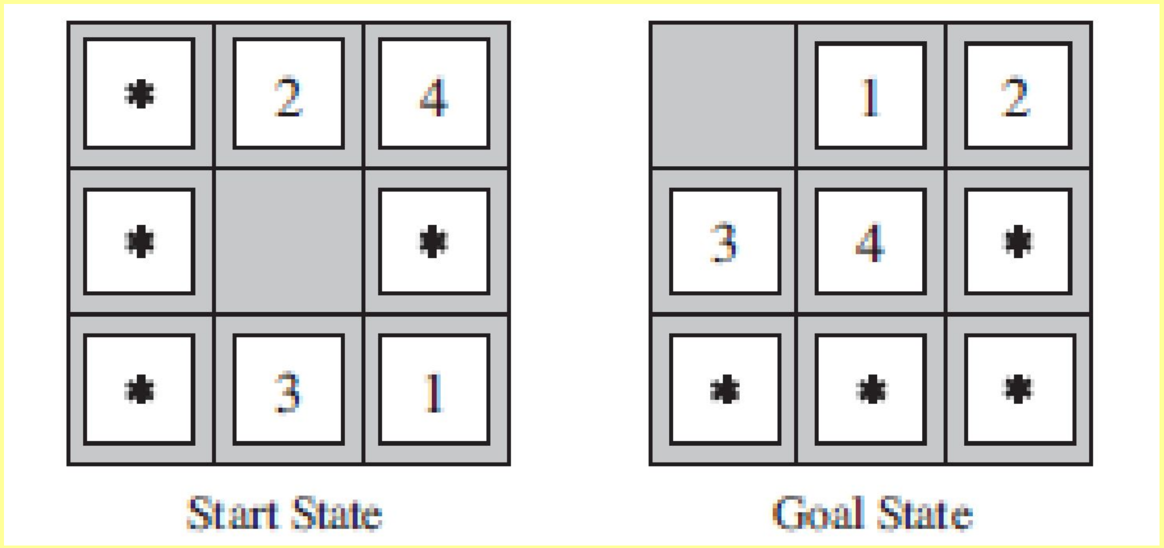
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution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

Generating admissible heuristics from relaxed problems

- If a collection of admissible heuristics h1 . . .hm is available for a problem and none of them dominates any of the others, which should we choose?
- As it turns out, we need not make a choice.

$$h(n) = \max\{h_1(n), \ldots, h_m(n)\}$$

Generating admissible heuristics from subproblems: Pattern databases



The idea behind **pattern databases** is to store these exact solution costs for every possible subproblem instance—in the example, every possible configuration of the four tiles and the blank.

Learning heuristics from experience

- How could an agent construct to estimate the cost of a solution beginning from the state at node n?
  - to devise relaxed problems
  - to learn from experience
  - Learning algorithm can be used to construct a function h(n) that can (with luck) predict solution costs for other states that arise during search. - neural nets, decision trees, reinforcement learning, inductive learning methods

End of unit 2

Next class – Logical Agents