Informed Search

Sanja Lazarova-Molnar

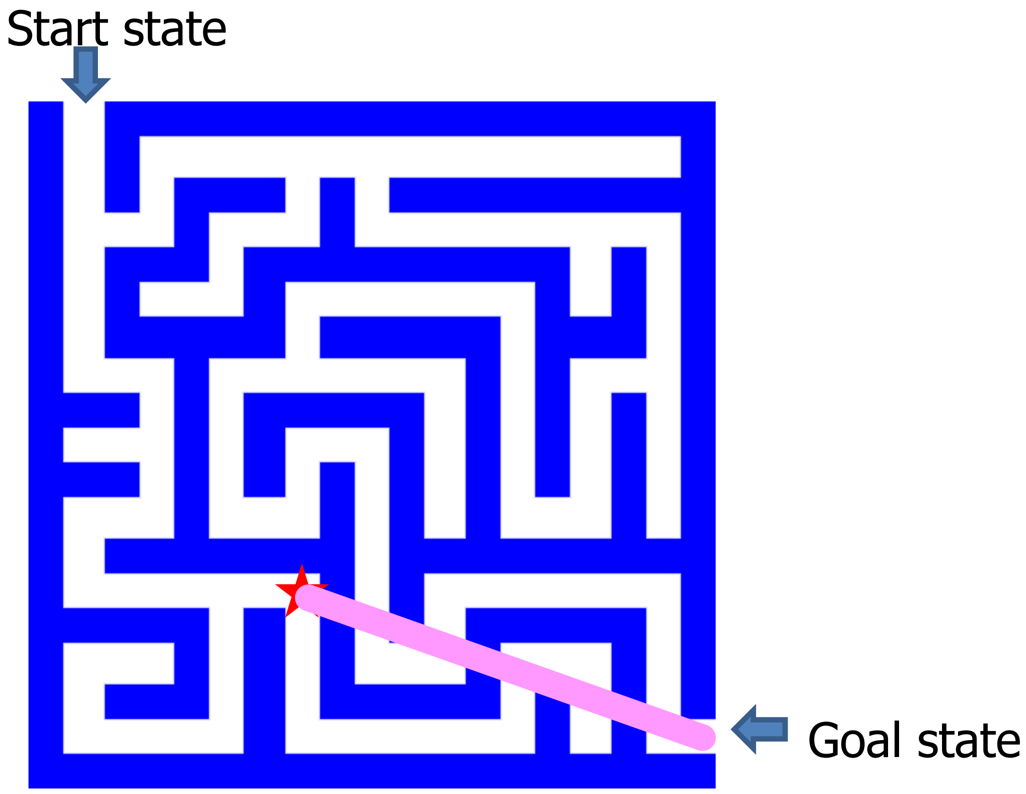
Informed search

- Idea: give the algorithm “hints” about the desirability of different states
  - Use an *evaluation function* to rank nodes and select the most promising one for expansion

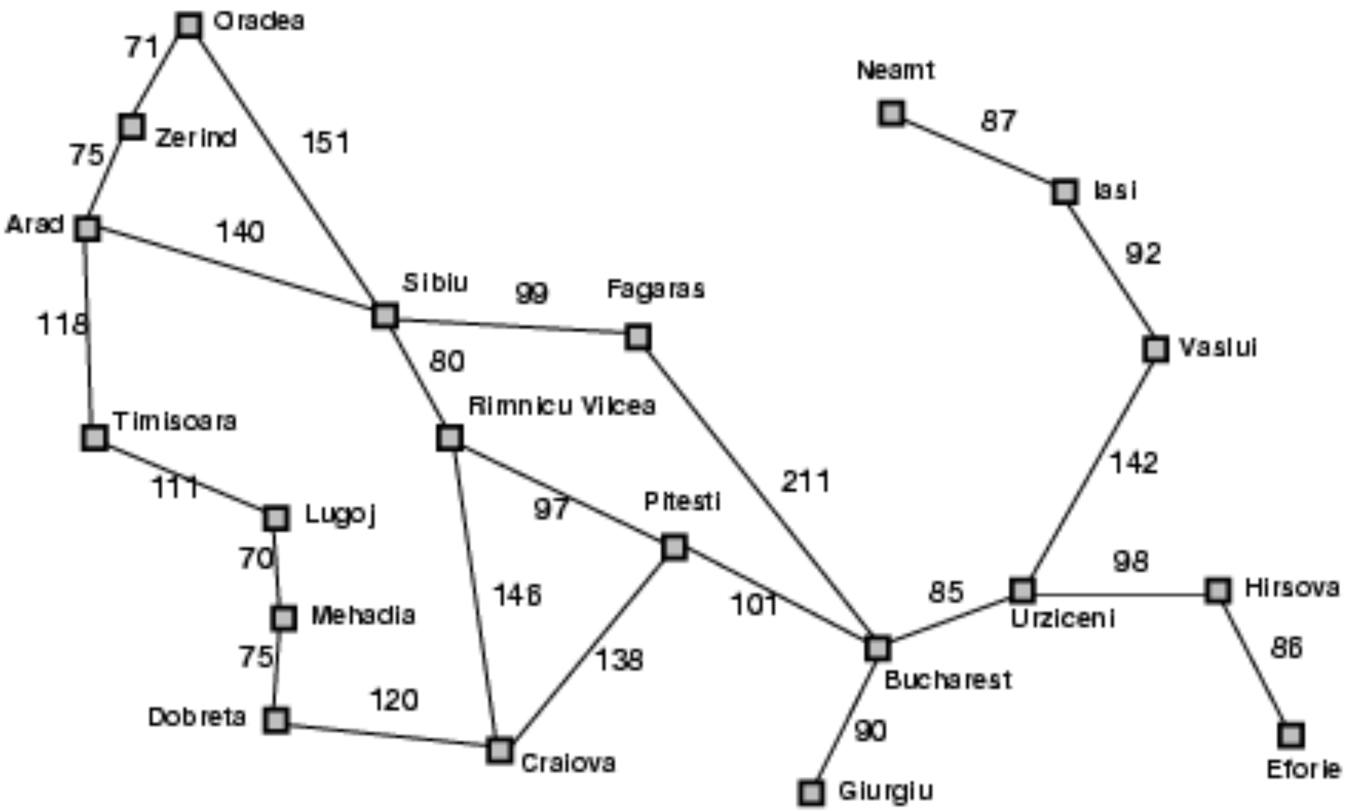
- Greedy best-first search
- A* search

Heuristic function

- Heuristic function $h(n)$ estimates the cost of reaching goal from node n
- Example:



Heuristic for the Romania problem



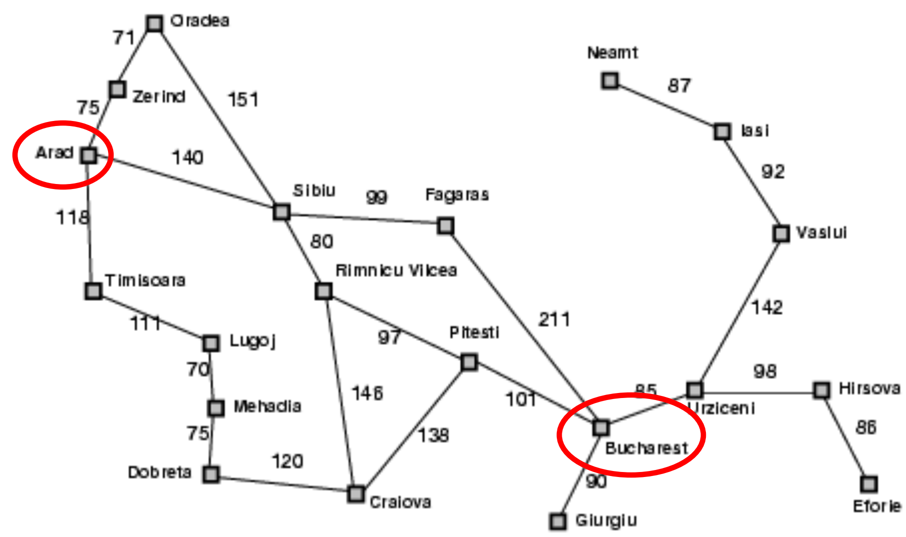
Straight–line distance to Bucharest

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

- Expand the node that has the lowest value of the heuristic function $h(n)$

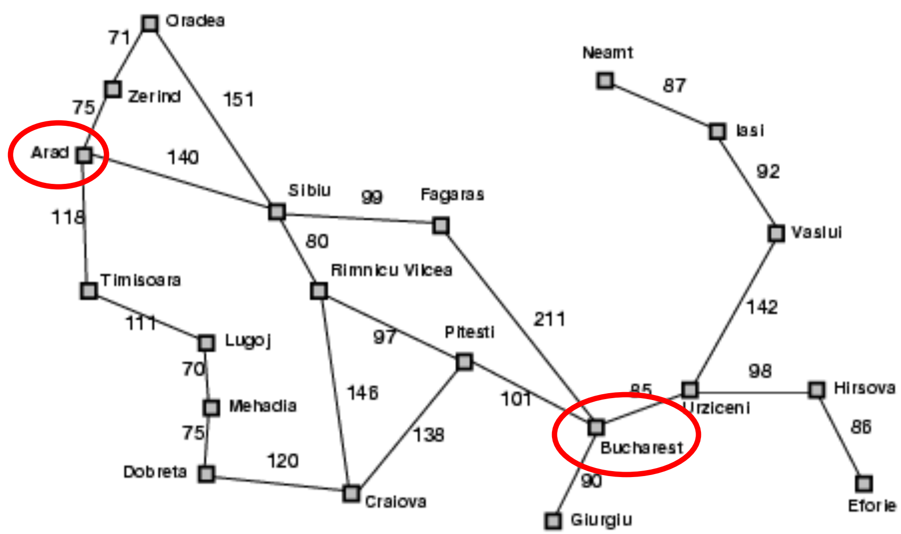
Greedy best-first search example



Straight–line distance to Bucharest

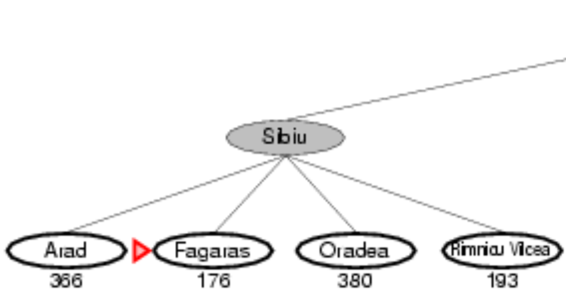
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

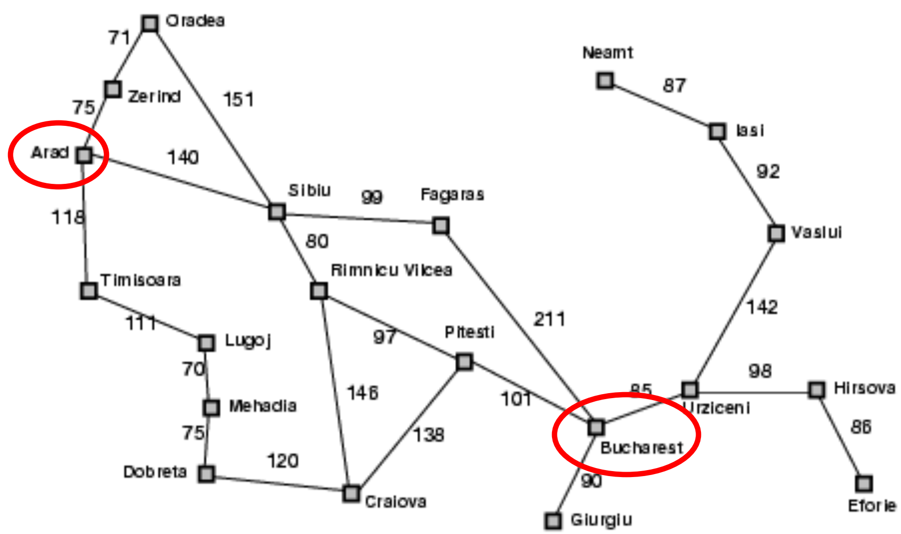
Greedy best-first search example



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

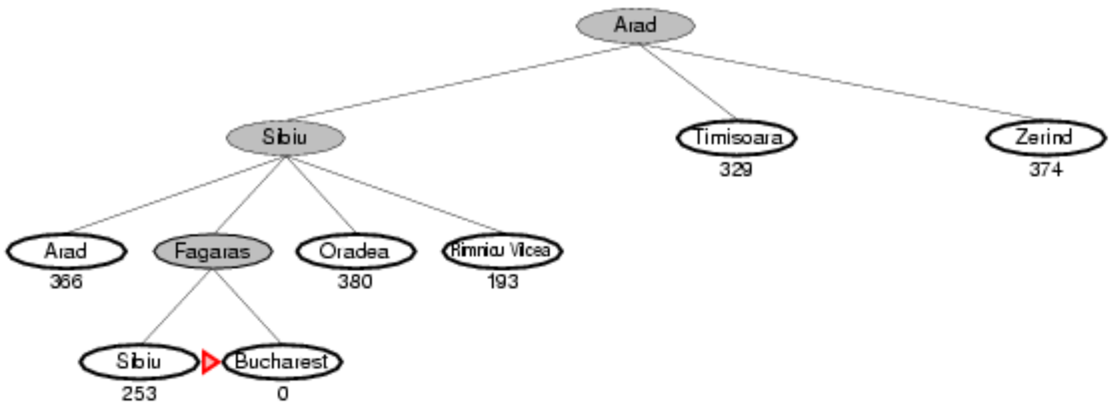


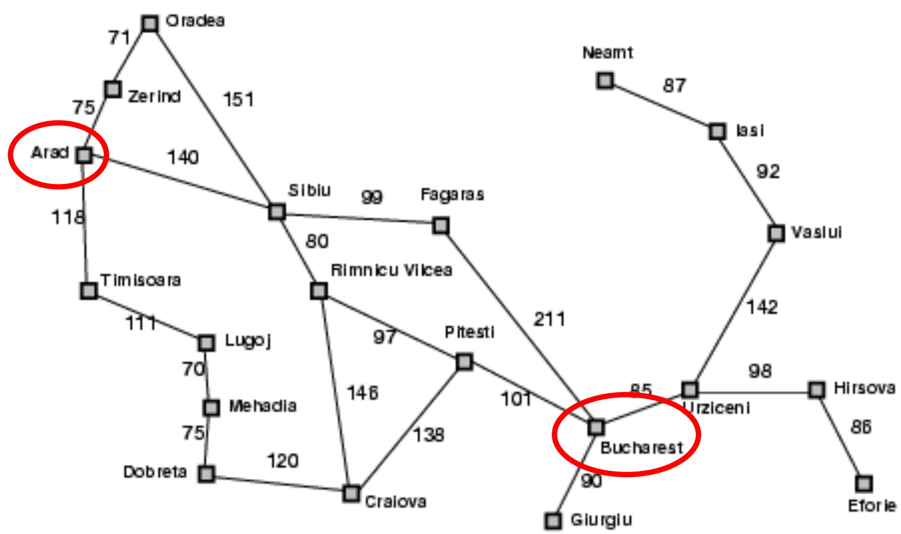


Straight–line distance
to Bucharest

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

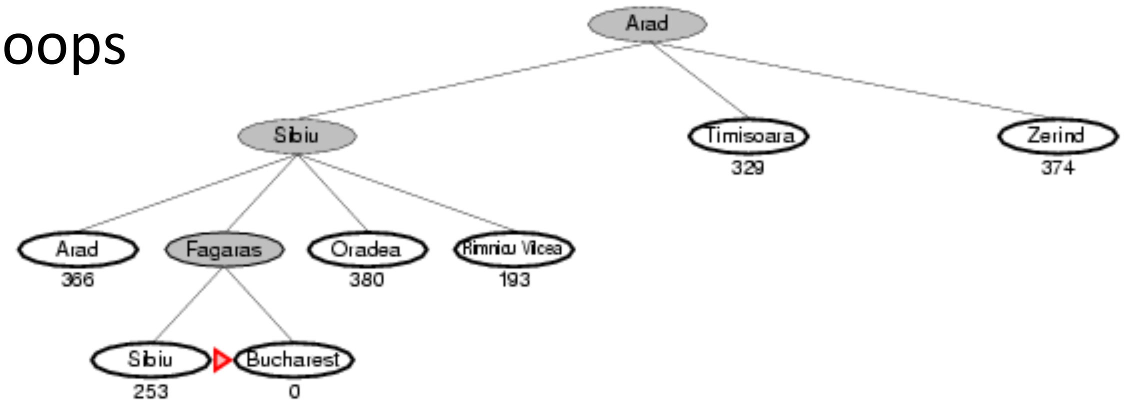
Properties of greedy best-first search

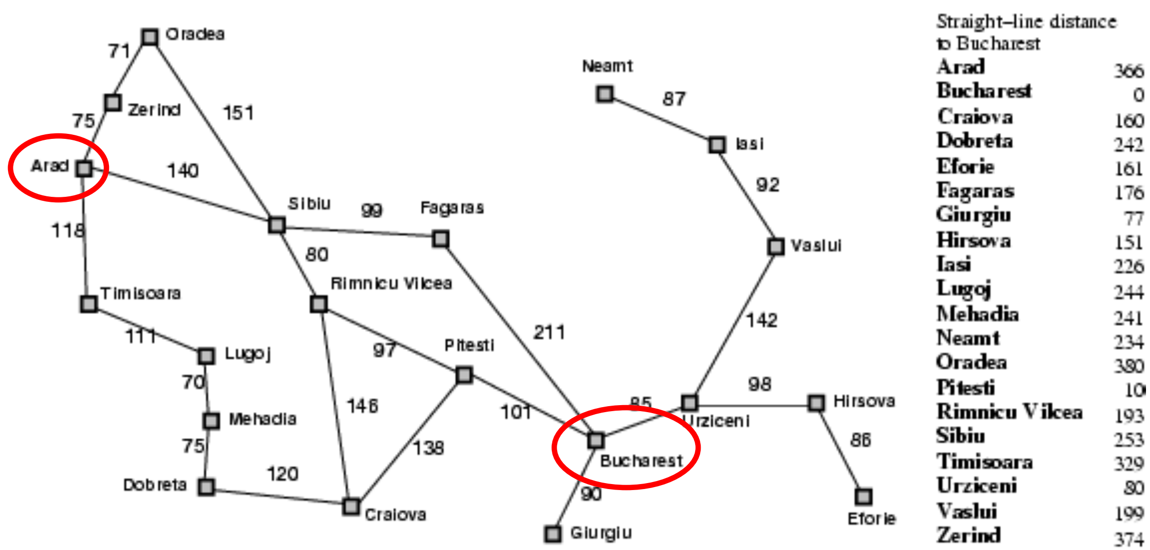
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops
- **Optimal?**

  No
- **Time?**

  Worst case: $O(b^m)$

  Best case: $O(bd)$ – If $h(n)$ is 100% accurate
- **Space?**

  Worst case: $O(b^m)$

How can we fix the greedy problem?

- Add another parameter to evaluate nodes!?

A^* search

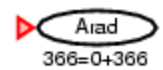
- Idea: avoid expanding paths that are already expensive
- The evaluation function $f(n)$ is the estimated total cost of the path through node n to the goal:

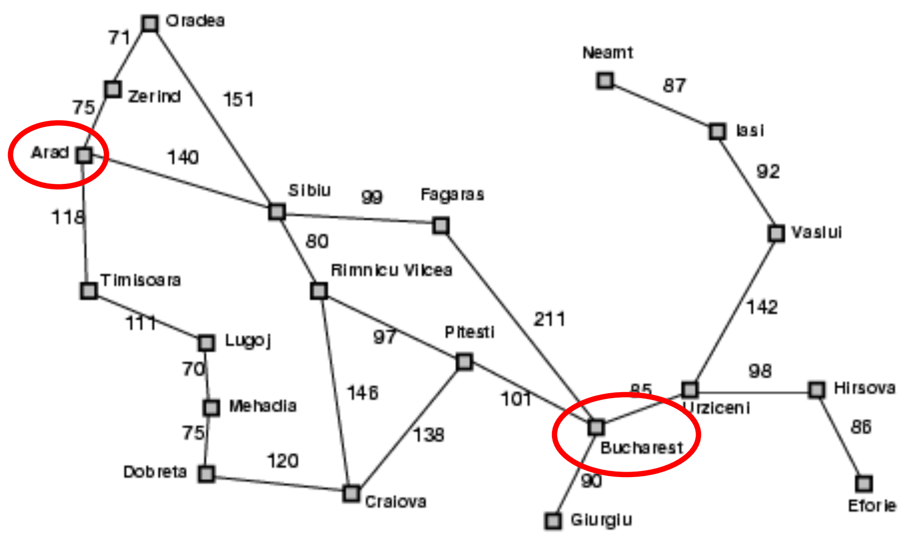
$$f(n) = g(n) + h(n)$$

$g(n)$: cost so far to reach n (path cost)

$h(n)$: estimated cost from n to goal (heuristic)

A* search example

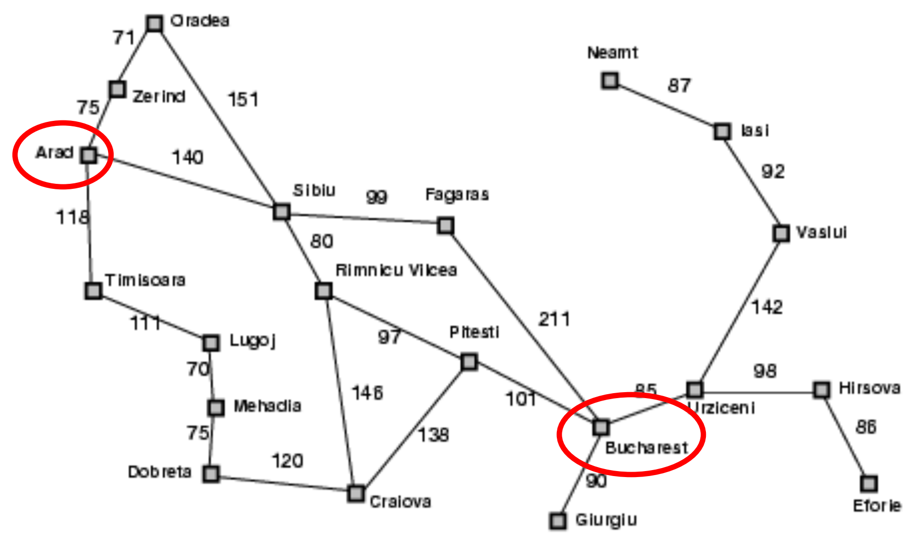




Straight−line distance
to Bucharest

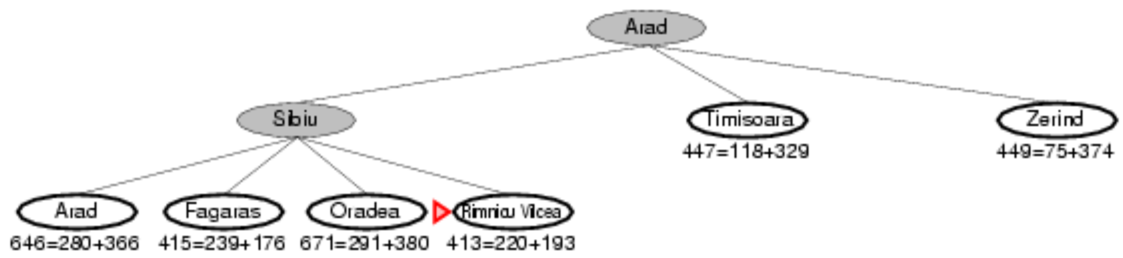
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

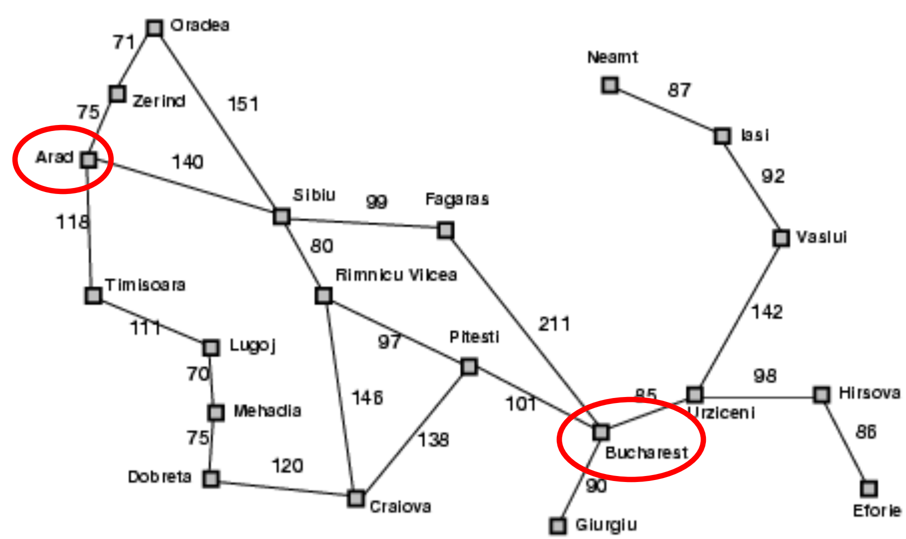
A* search example



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

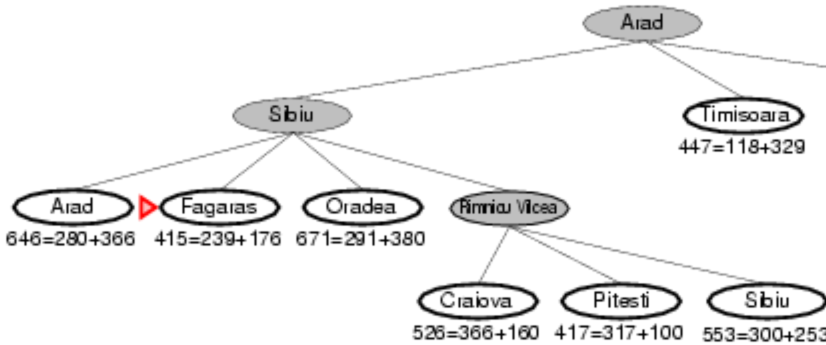


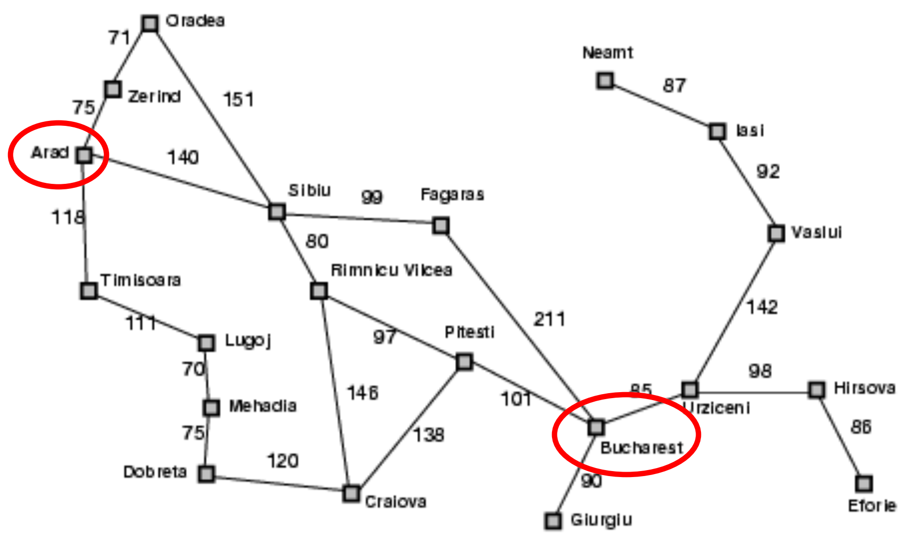


Straight–line distance to Bucharest

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[illegible]
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

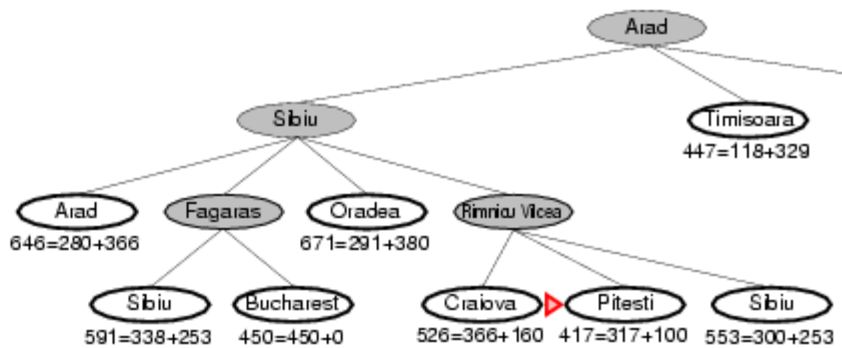


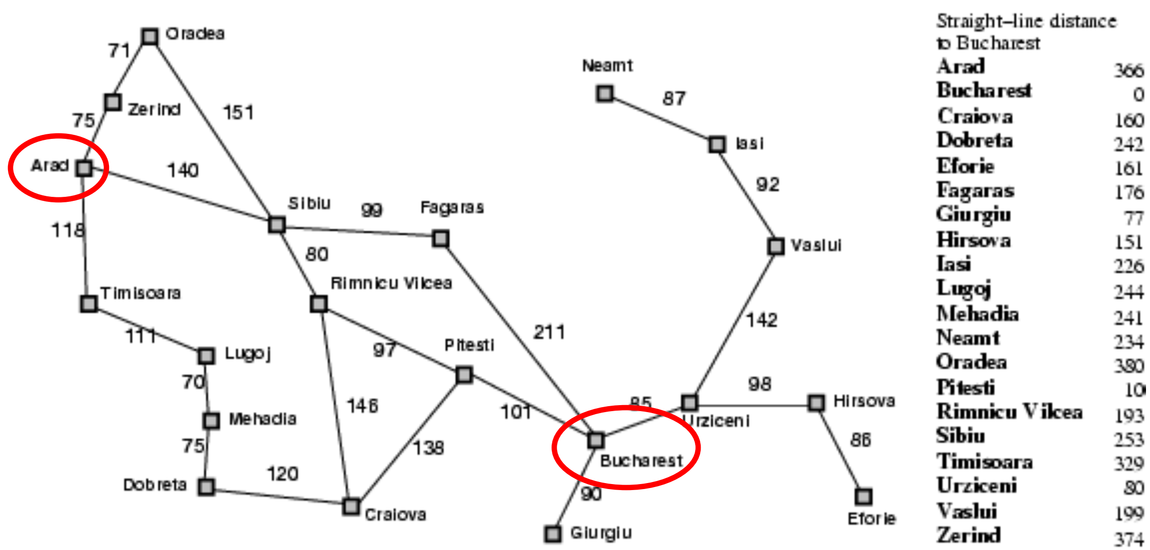


Straight–line distance to Bucharest

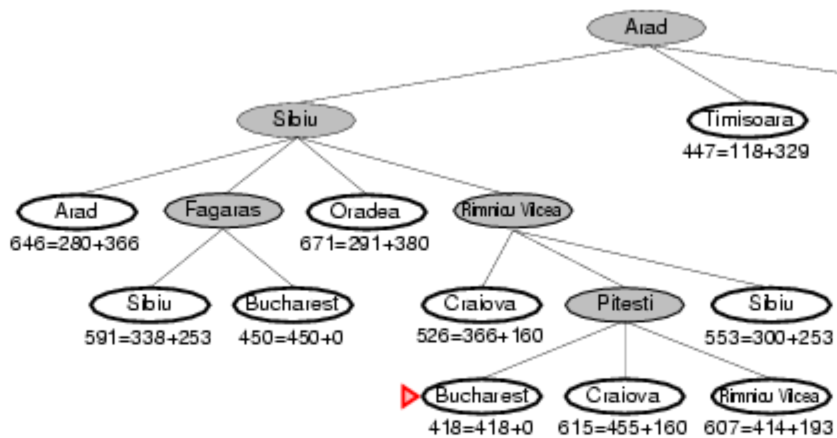
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

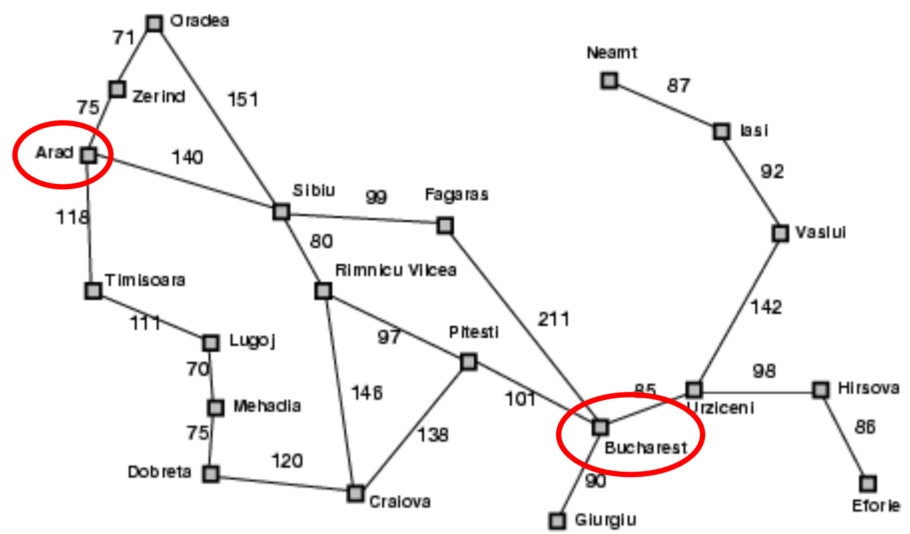
A* search example





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

Admissible heuristics

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- 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
- Example: straight line distance never overestimates the actual road distance
- **Theorem:** If $h(n)$ is admissible, A^* is optimal

Optimality of A*

- A* is *optimally efficient* – no other tree-based algorithm that uses the same heuristic can expand fewer nodes and still be guaranteed to find the optimal solution
  - any algorithm that does not expand all nodes in the contours between the root and the goal contour runs the risk of missing the optimal solution

Properties of A*

- **Complete?**

  Yes – unless there are infinitely many nodes with $f(n) \leq C^*$
- **Optimal?**

  Yes
- **Time?**

  Number of nodes for which $f(n) \leq C^*$ (exponential)
- **Space?**

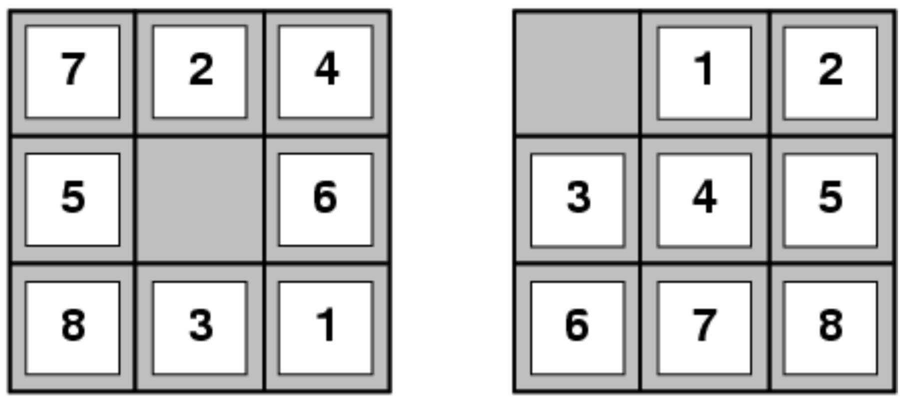
  Exponential

Designing heuristic functions

- Heuristics for the 8-puzzle

  $h_1(n)$ = number of misplaced tiles

  $h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)



Start State Goal State

$$h_1(\text{start}) = 8$$

$$h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18$$

- Are h_1 and h_2 admissible?

Dominance

- If h_1 and h_2 are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all n, (both admissible) then h_2 dominates h_1
- Which one is better for search?
  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  - Therefore, A* search with h_1 will expand more nodes, so h_2 is better

C^* - optimal cost

Heuristics from 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

- Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths):

- *d*=12 IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes

- *d*=24 IDS ≈ 54,000,000,000 nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Combining heuristics

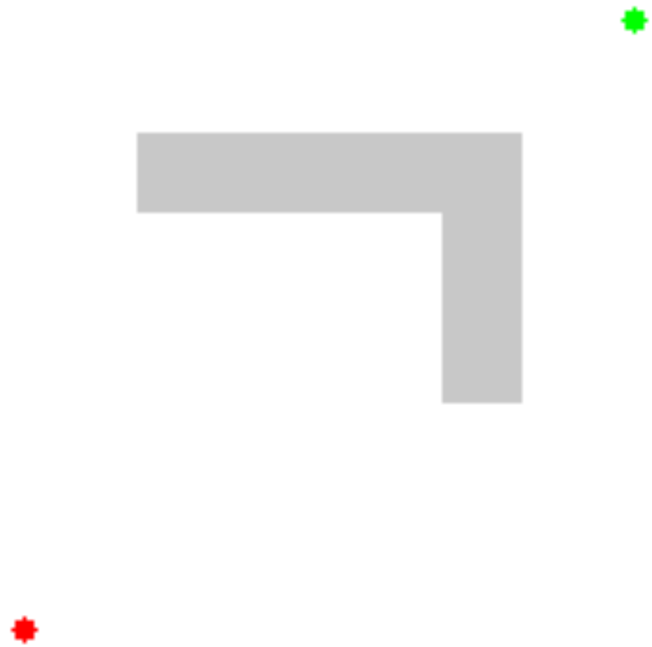
- Suppose we have a collection of admissible heuristics $h_1(n), h_2(n), \ldots, h_m(n)$, but none of them dominates the others
- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

Weighted A* search

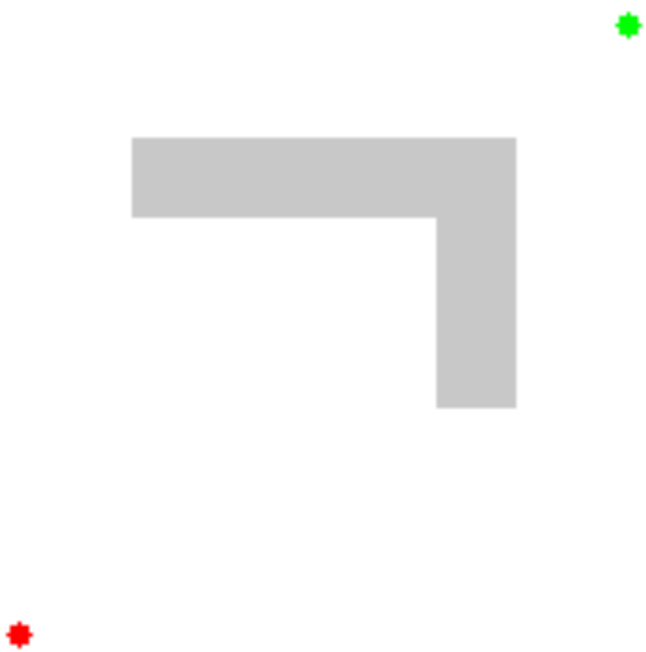
- Idea: speed up search at the expense of optimality
- Take an admissible heuristic, "inflate" it by a multiple $\alpha > 1$, and then perform A* search as usual
- Fewer nodes tend to get expanded, but the resulting solution may be suboptimal (its cost will be at most α times the cost of the optimal solution)

Example of weighted A* search

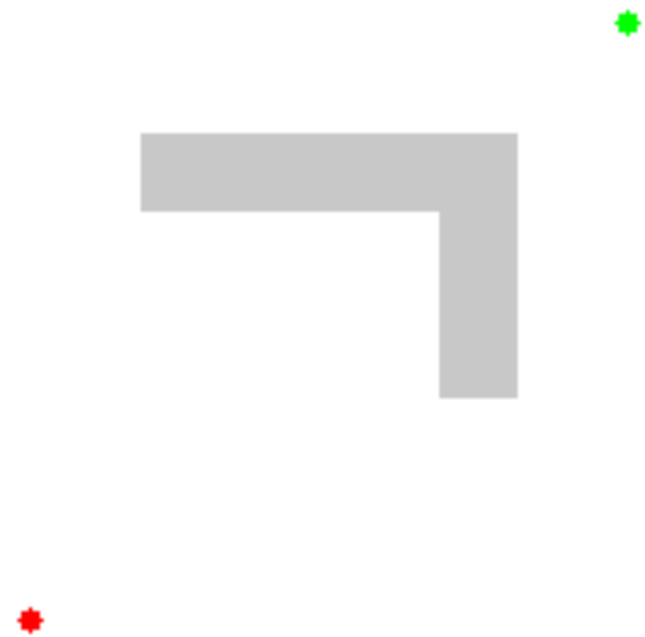


Heuristic: 5 * Euclidean distance from goal
Source: Wikipedia

Example of weighted A* search



Heuristic: 5 * Euclidean distance from goal
Source: Wikipedia



Compare: Exact A*

Memory-bounded search

- The memory usage of A* can still be exorbitant
- How to make A* more memory-efficient while maintaining completeness and optimality?

- IDA* (Iterative Deepening A*)
- SMA* (Simplified Memory Bounded A*)
  - Forget some subtrees but remember the best f-value in these subtrees and regenerate them later if necessary

- Problems: memory-bounded strategies can be complicated to implement, suffer from "thrashing"
  - repeated pruning and regeneration of the same few nodes

SMA*

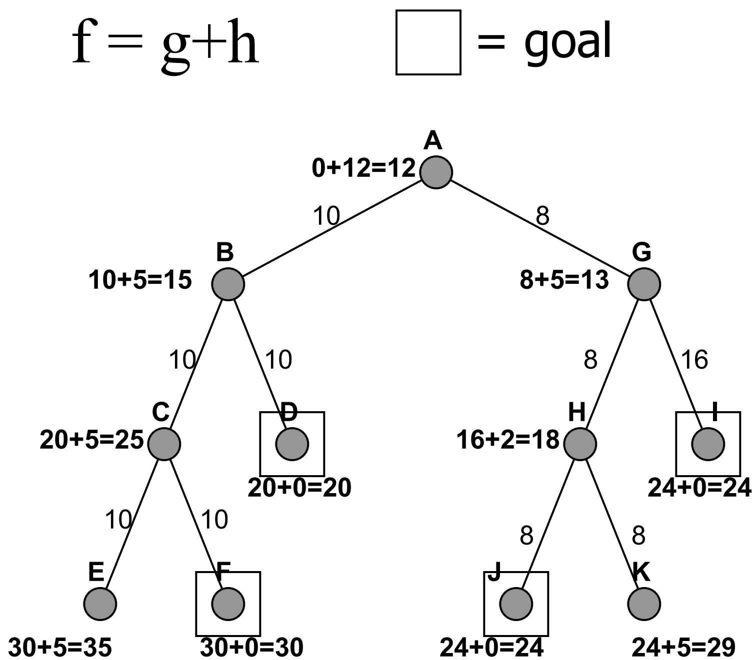
- Optimizes A* to work within reduced memory
- Key Idea:
  - IF memory full for extra node
  - Remove highest f-value leaf
  - Remember best-forgotten child in each parent node
- Generate Children 1 by 1
  - Expanding: add <u>1 child at a time</u> to QUEUE – Avoids memory overflow
  - Allows monitoring if nodes need deletion
- Too long paths: Give up
  - **Extending** path **cannot fit** in **memory**: give up
- Set **f-value** node to infinity
  - **Remembers:** path cannot be found here

SMA*

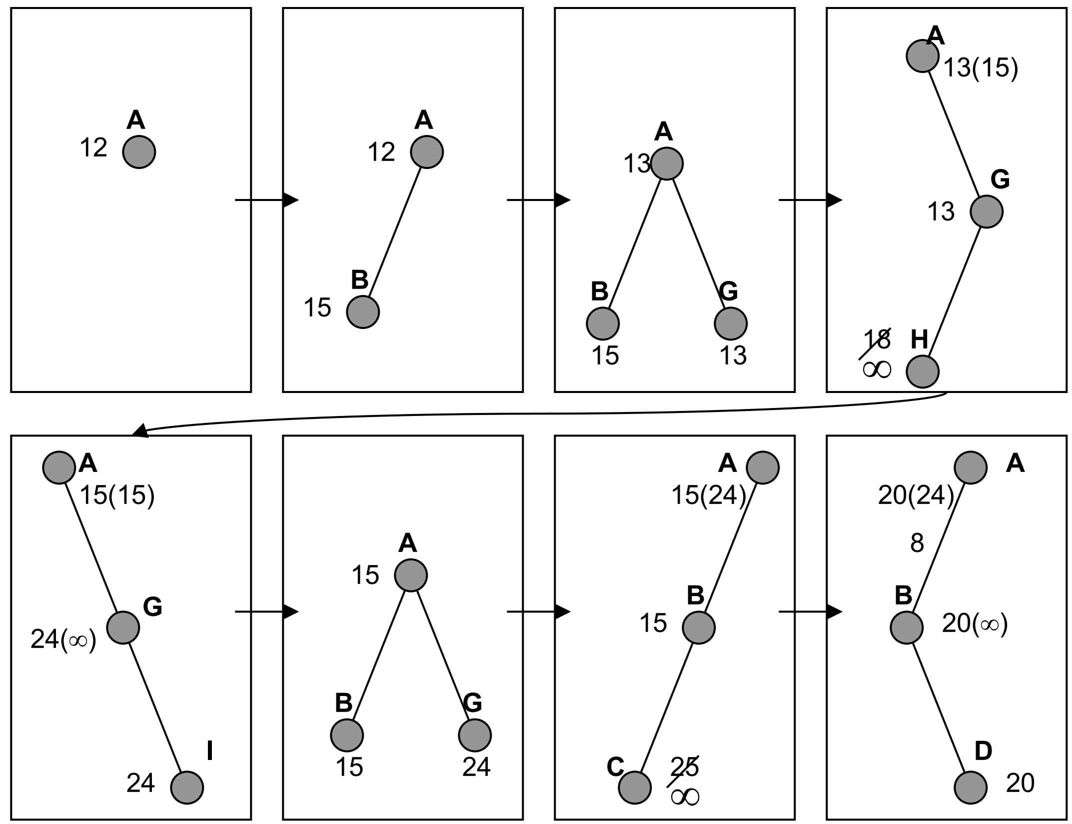
- Adjust f-values
  - IF all children M_i of node N have been explored
  - AND for all i: $f(M_i) > f(N)$
  - THEN reset (through N means through children)
    - $f(N) = \min\{f(M_i) \mid M_i \text{ child of } N\}$

Simple Memory-bounded A* (SMA*)

Search *space*



Progress of SMA* (with enough memory to store just 3 nodes).
Each node is labeled with its *current f*-cost.
Values in parentheses show the value of the best forgotten descendant (child node).



Optimal & complete if enough memory
Can be made to signal when the best solution found might not be optimal (e.g., if J=19)

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  local variables: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
         f(s) ← ∞
     else
         f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
         update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
         delete shallowest, highest-f-cost node in Queue
         remove it from its parent's successor list
         insert its parent on Queue if necessary
     insert s on Queue
  end
```

Figure 4.12 Sketch of the SMA* algorithm. Note that numerous details have been omitted in the interests of clarity.

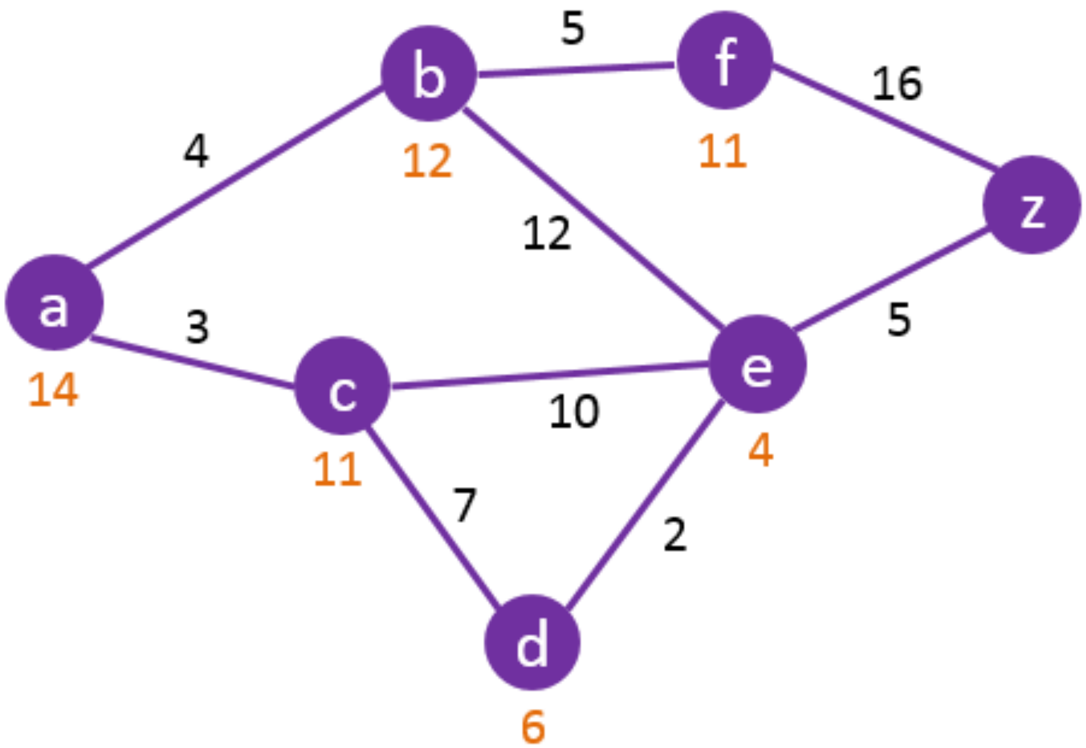
Uninformed search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$

b: maximum branching factor of the search tree
d: depth of the optimal solution
m: maximum length of any path in the state space
C^*: cost of optimal solution

All search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes	Number of nodes with $g(n)+h(n) \leq C^*$	



A* Search Algorithm

What is the shortest path to travel from A to Z?

Numbers in orange are the heuristic values, distances in a straight line (as the crow flies) from a node to node Z.