Last update: September 11, 2012

Informed search algorithms

CMSC 421: Chapter 3, Sections 5–6

Motivation

◇ In the previous sections we talked about trial-and-error search

◇ In the worst case, most searches take exponential time (unless P=NP)

◇ Can sometimes do much better on the average, using *heuristics*

◇ *Heuristic:*

- Rule of thumb, simplification, or educated guess
- Reduces the search for solutions in domains that are difficult and poorly understood

◇ Depending on what heuristic you use, you won't necessarily find an optimal solution, or even a solution at all

◇ We'll look at some conditions under which we're guaranteed to find optimal solutions

Heuristic tree search

```
function CAUTIOUS-TREE-SEARCH(problem)      # like TREE-SEARCH in the book
    frontier ← list that contains a node for problem's initial state
    loop
        if frontier is empty then return Failure
        choose and remove a node x from frontier
        if STATE[x] is a goal then return the corresponding solution
        else expand x and add the new nodes to frontier
```

- ◊ Heuristic choice in search algorithms: **what node to expand next**
  - Use an *evaluation function* $f(x)$ for each node x
    - ◊ estimate of how much it will cost to get to a goal
- ◊ Always choose a frontier node with the lowest f value

Heuristic graph search

function GRAPH-SEARCH(*problem*)

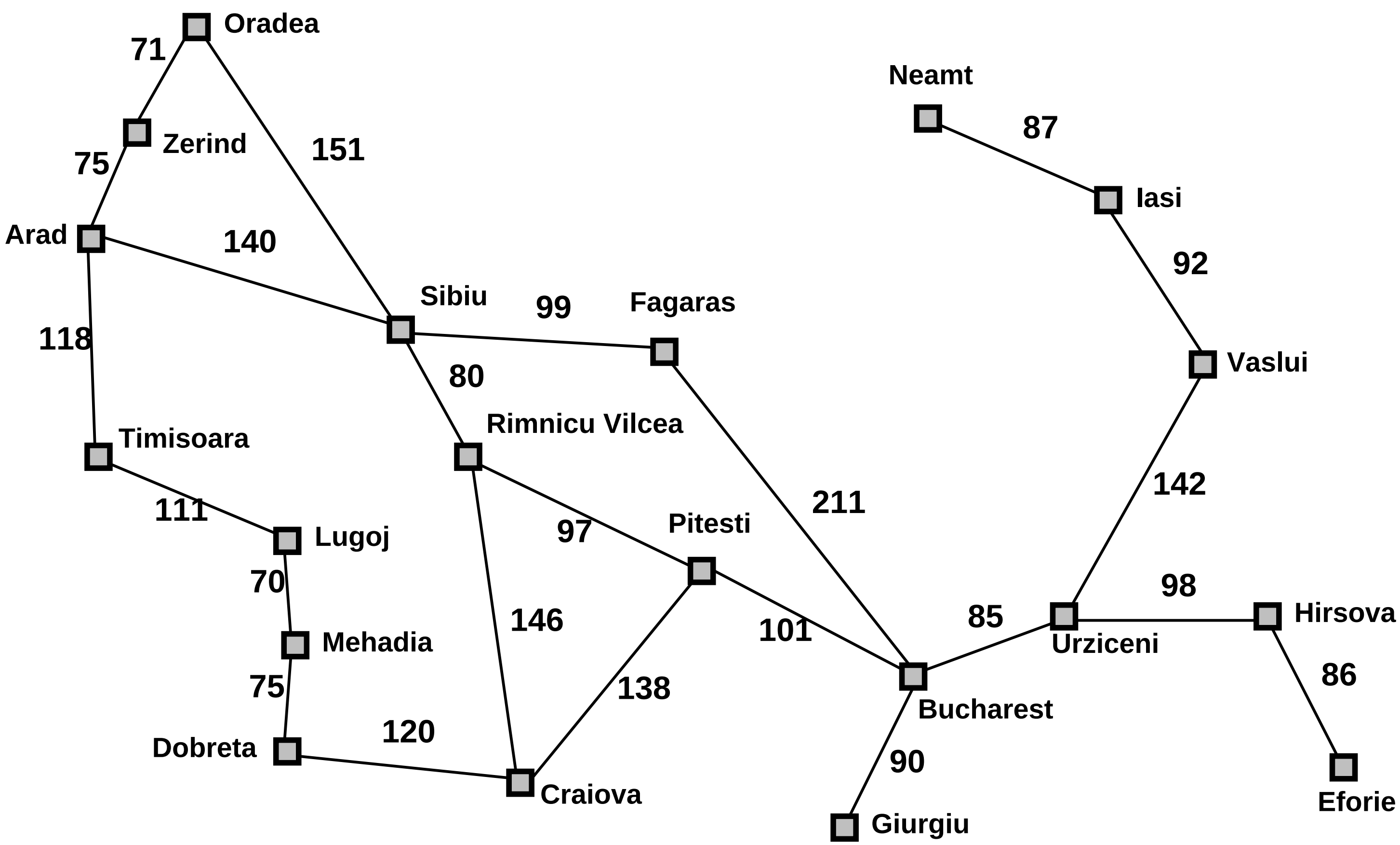
    frontier ← list that contains a node for *problem*'s initial state
    explored ← empty set
    loop
        if *frontier* is empty **then return** Failure
        choose and remove a node x from *frontier*
        if STATE[x] is a goal **then return** the corresponding solution
        if STATE[x] is not in *explored* **then**
            add STATE[x] to *explored*
            expand x and add the new nodes to *frontier*

◇ Again, always choose a frontier node with the lowest f value

Romania with step costs in km

◇ Recall that we want to get to Bucharest

◇ One possible heuristic: straight-line distance to Bucharest



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

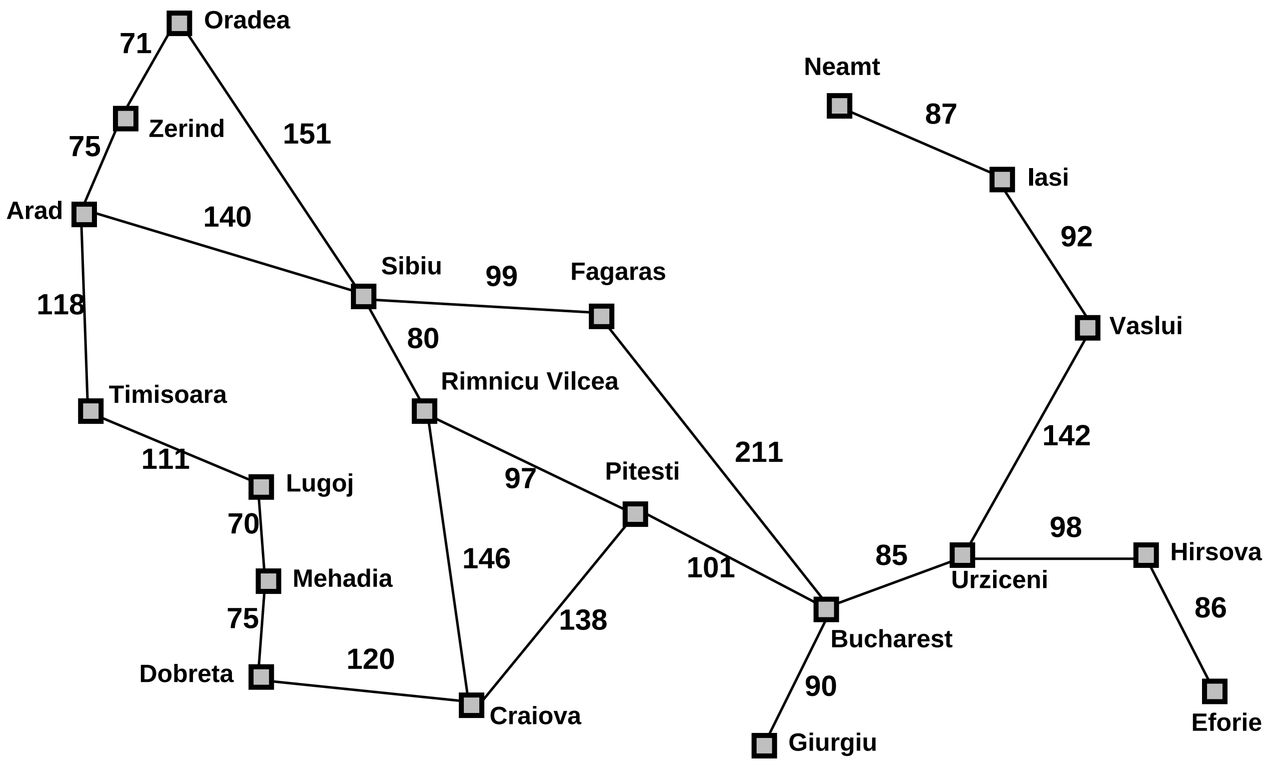
Greedy search

◊ *Heuristic function* $h(x)$ = estimate of cost from x to the closest goal

- E.g., $h_{\mathrm{SLD}}(x)$ = straight-line distance from x to Bucharest

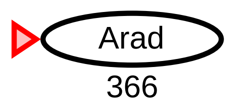
◊ Greedy search uses $f(x) = h(x)$,

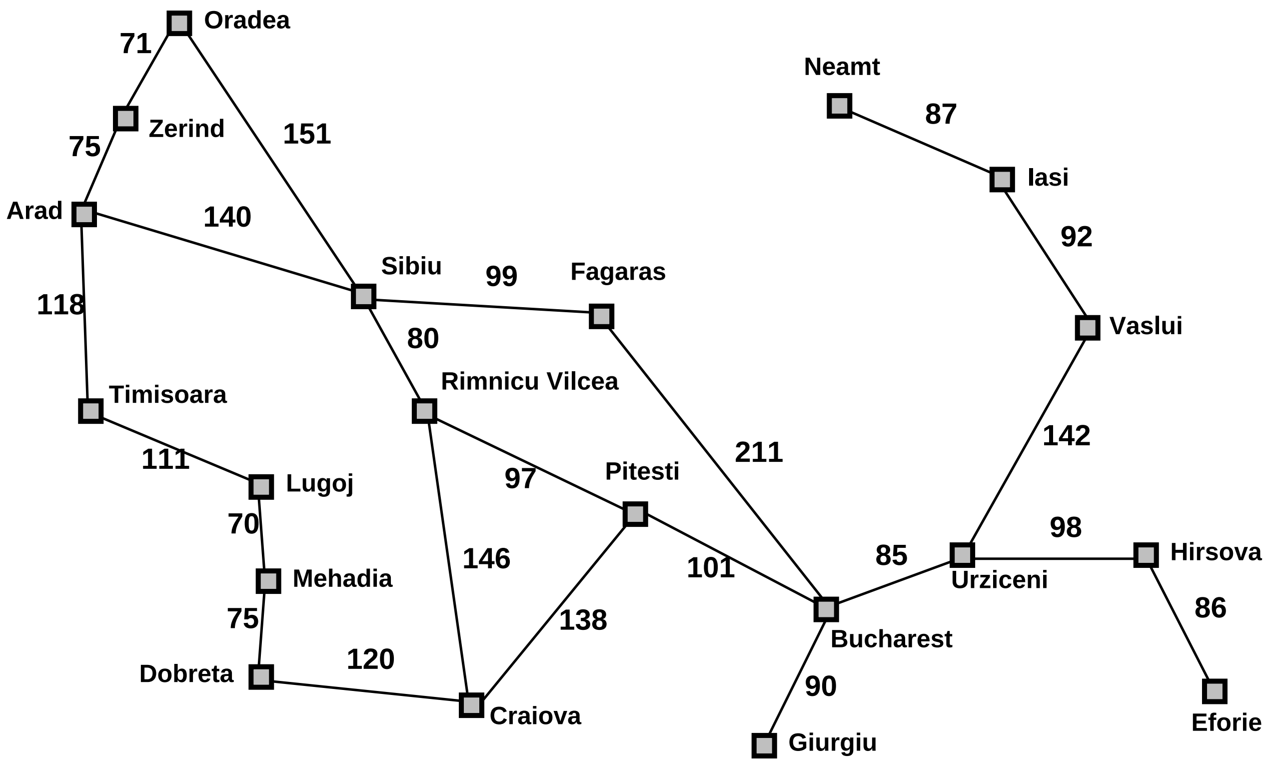
- keeps *frontier* ordered in increasing value of h
- always expands whatever node **appears** to be closest to a goal



Straight−line distance
to Bucharest

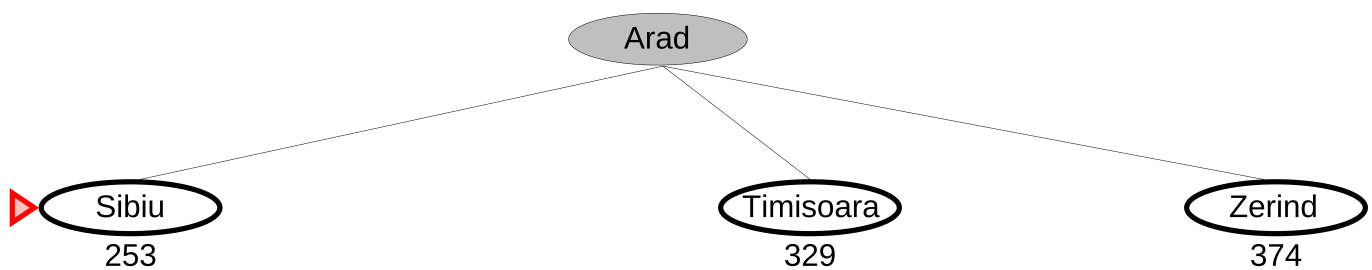
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

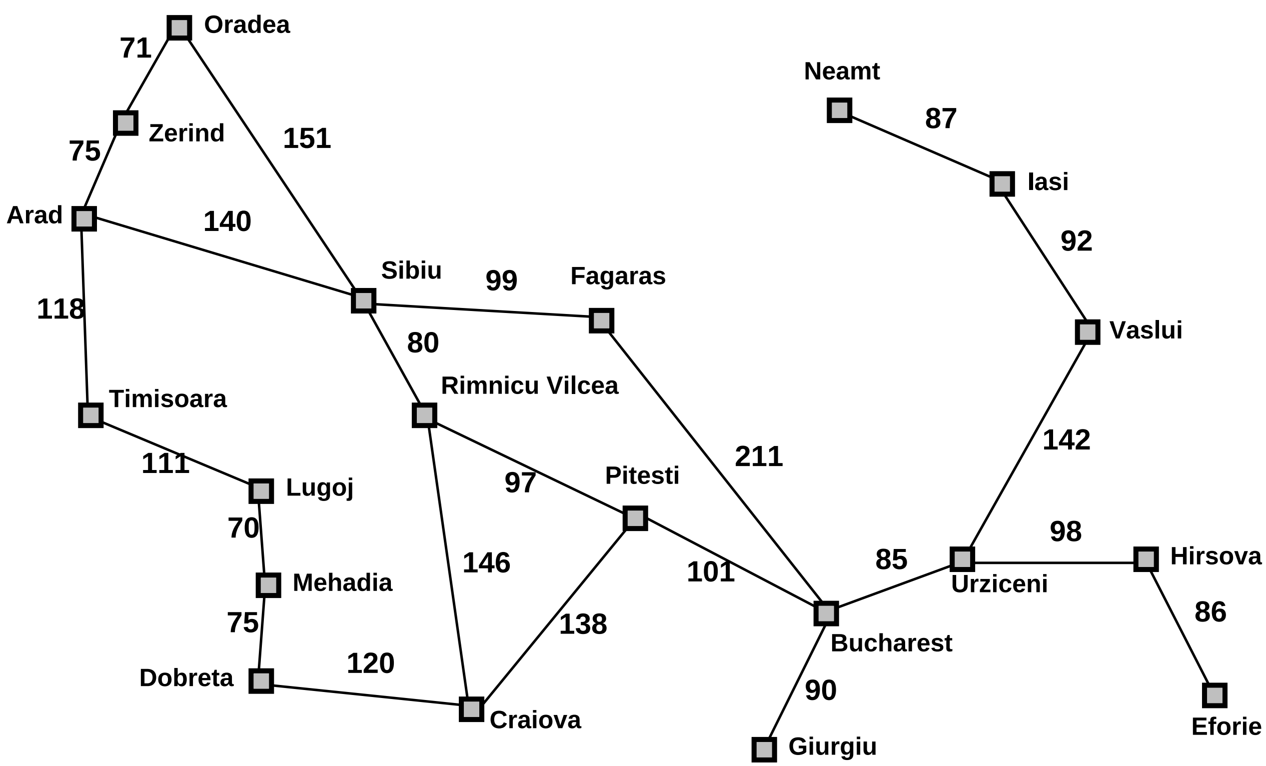




Straight−line distance
to Bucharest

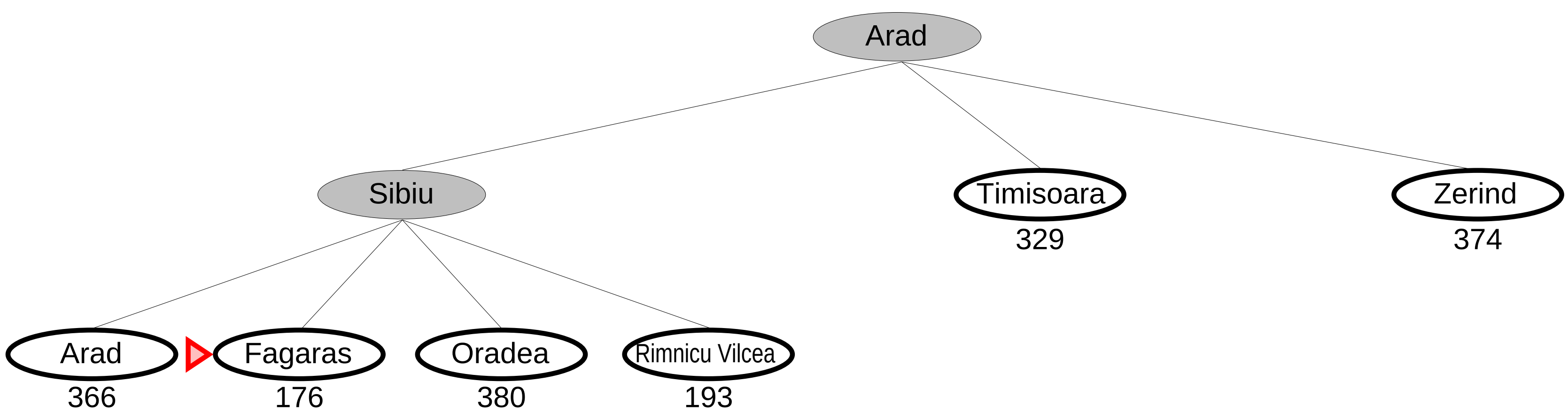
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

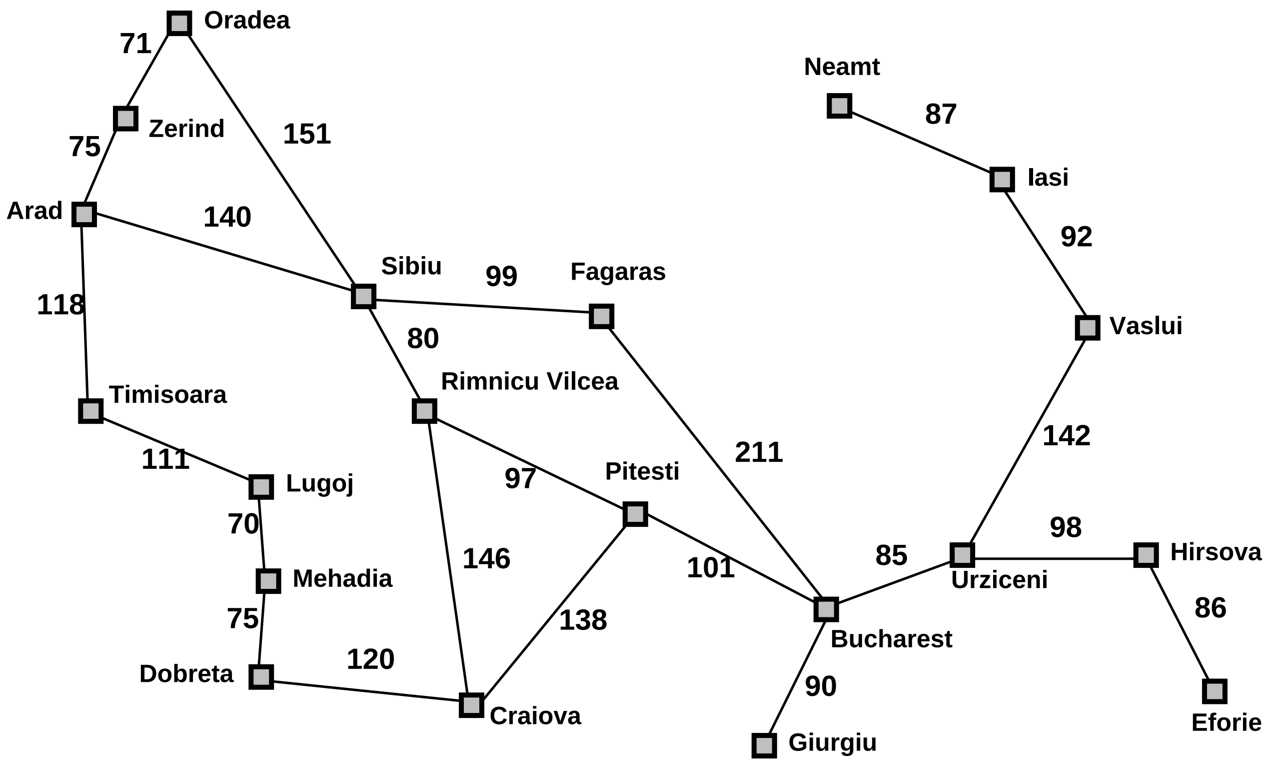




Straight−line distance
to Bucharest

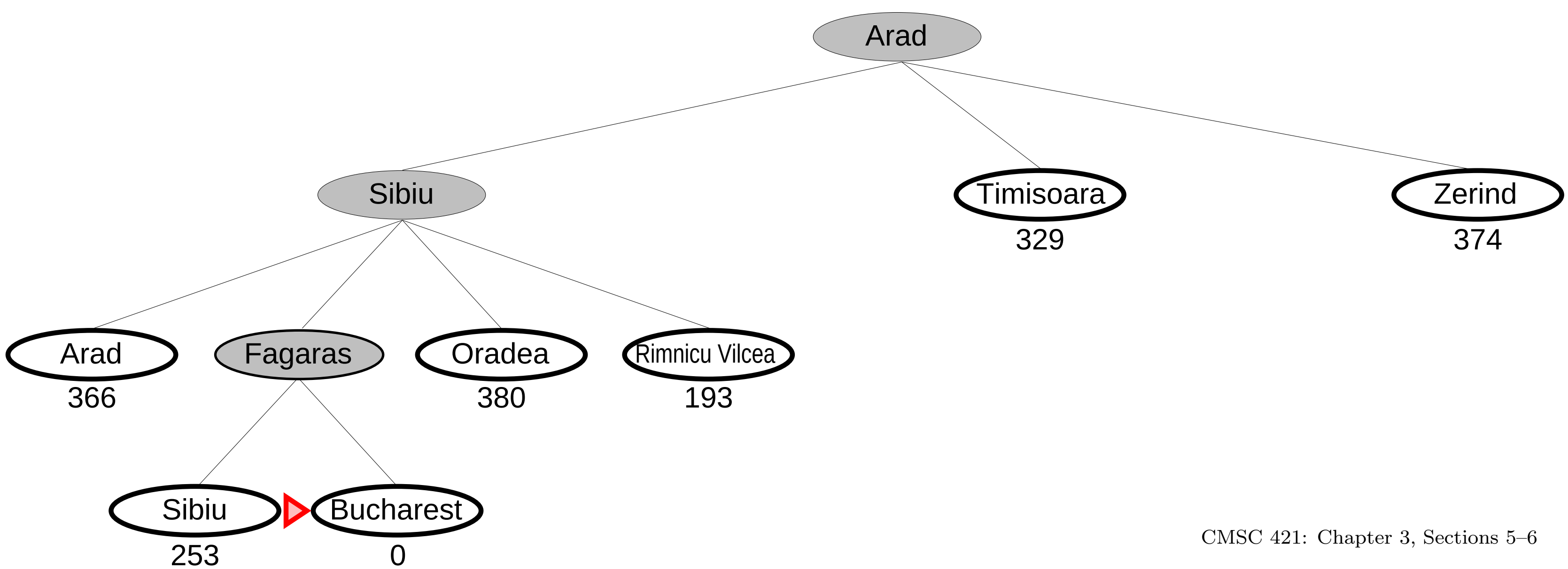
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





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



Properties of greedy search

◇ *Complete?*

Properties of greedy search

◇ *Complete?* No. Can get stuck in loops:
- Iasi → Neamt → Iasi → Neamt →
- Complete if search space is finite and we do loop-checking

◇ *Time?*

Properties of greedy search

◊ *Complete?* No. Can get stuck in loops:

- Iasi → Neamt → Iasi → Neamt →
- Complete if search space is finite and we do loop-checking

◊ *Time?* $O(b^m)$, but a good heuristic can give a **huge** improvement

◊ *Space?*

Properties of greedy search

◊ *Complete?* No. Can get stuck in loops:

- Iasi → Neamt → Iasi → Neamt →
- Complete if search space is finite and we do loop-checking

◊ *Time?* $O(b^m)$, but a good heuristic can give a **huge** improvement

◊ *Space?* $O(b^m)$—keeps all nodes in memory

◊ *Optimal?*

Properties of greedy search

◇ *Complete?* No. Can get stuck in loops:
- Iasi → Neamt → Iasi → Neamt →
- Complete if search space is finite and we do loop-checking

◇ *Time?* $O(b^m)$, but a good heuristic can give a **huge** improvement

◇ *Space?* $O(b^m)$—keeps all nodes in memory

◇ *Optimal?* No

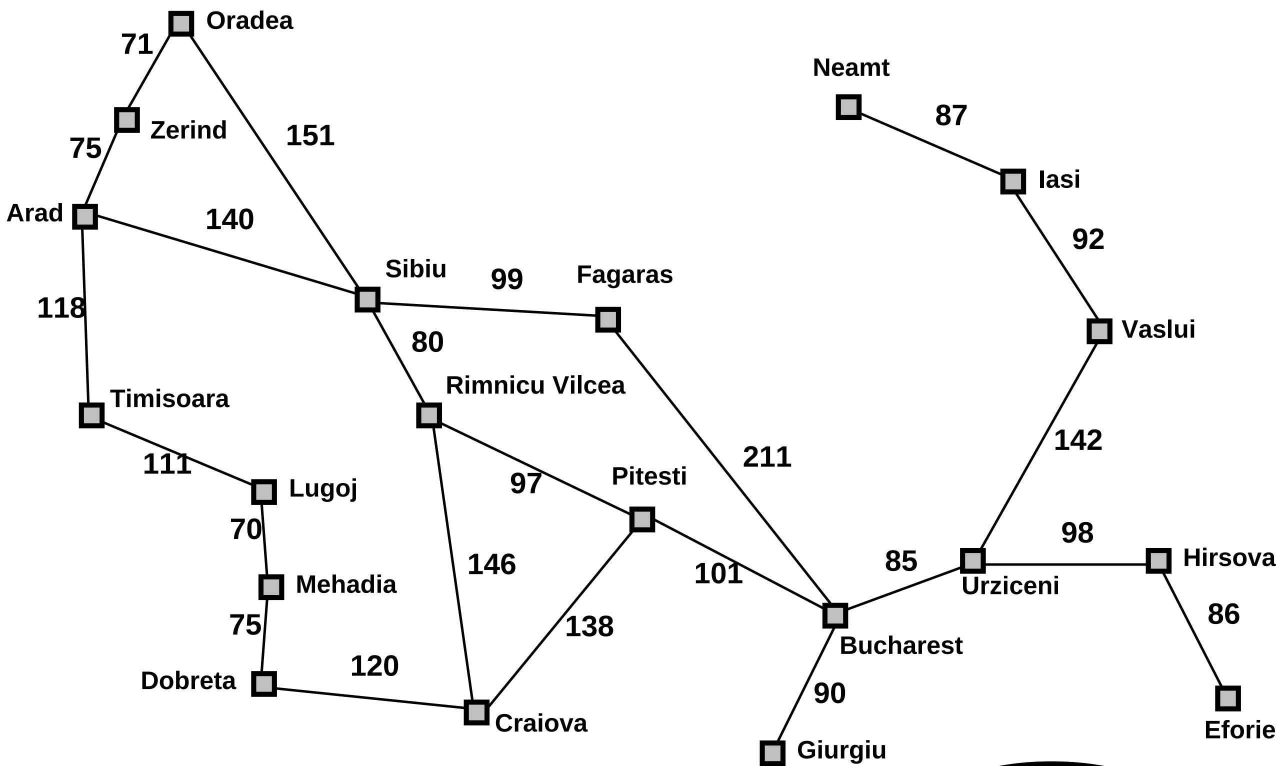
◇ A **greedy search** isn't the same as an ordinary **greedy algorithm**
- A greedy algorithm doesn't remember all of *frontier*
  - Just the current path, never goes back to look at others
  - Runs in time $O(l)$ if it finds a solution of length l
- Loop-checking cannot make it complete
  - Choose a path that doesn't lead to a solution $\Longrightarrow$ will fail
  - Normally used on problems where all paths lead to solutions

A^* tree search

◇ Idea: avoid expanding paths that are already expensive

◇ Evaluation function $f(x) = g(x) + h(x)$

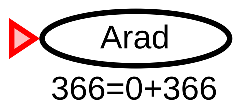
- $g(x)$ = cost to reach node x
- $h(x)$ = estimated cost from x to goal
- $f(x)$ = estimated total cost of path through x to goal

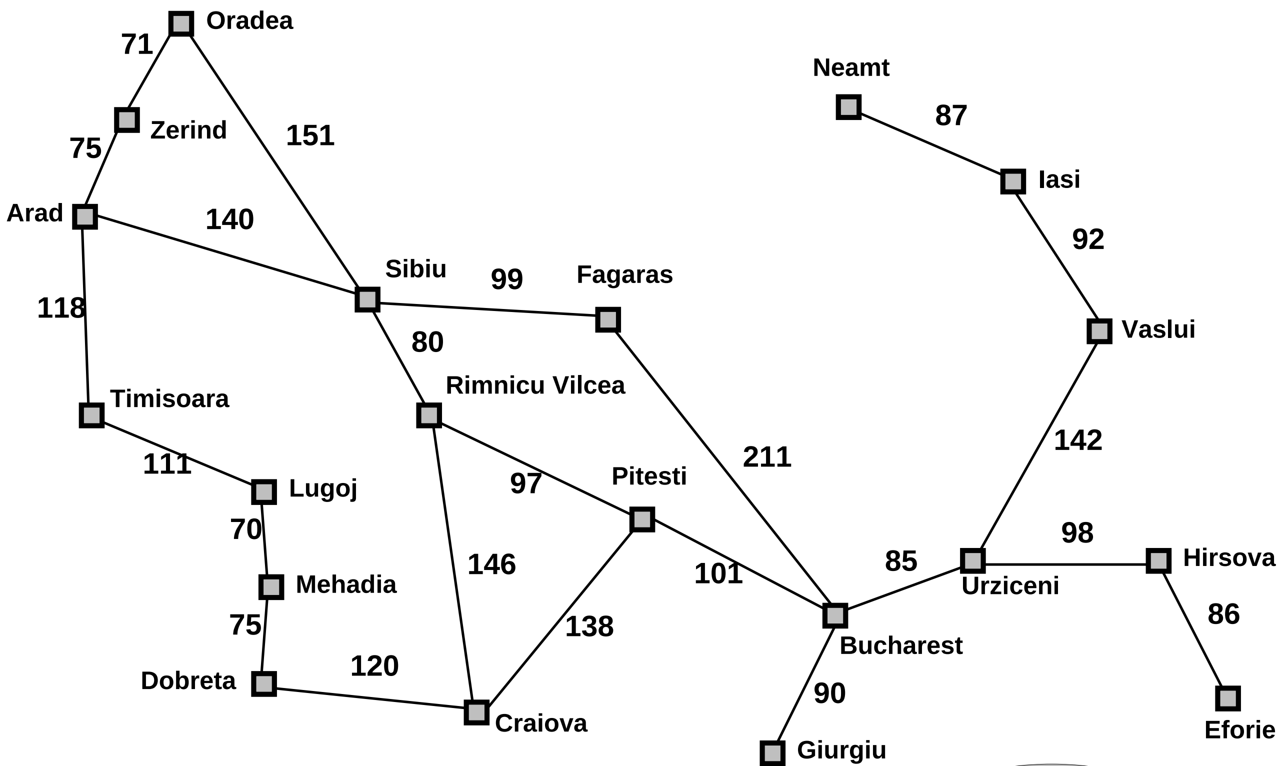


Straight−line distance to Bucharest

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

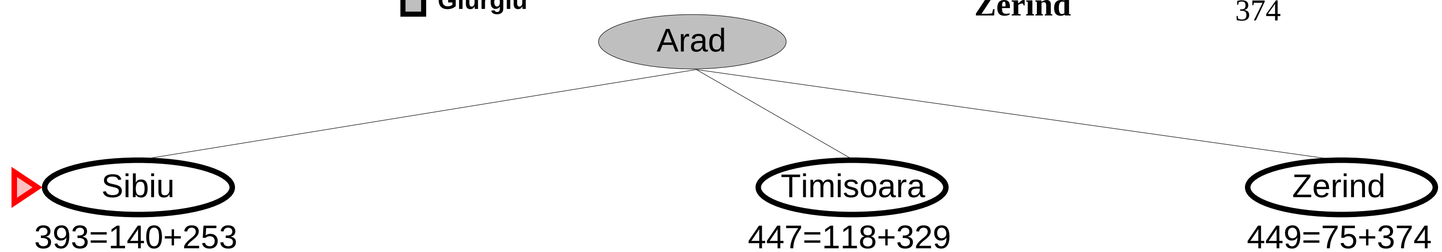
Search tree:

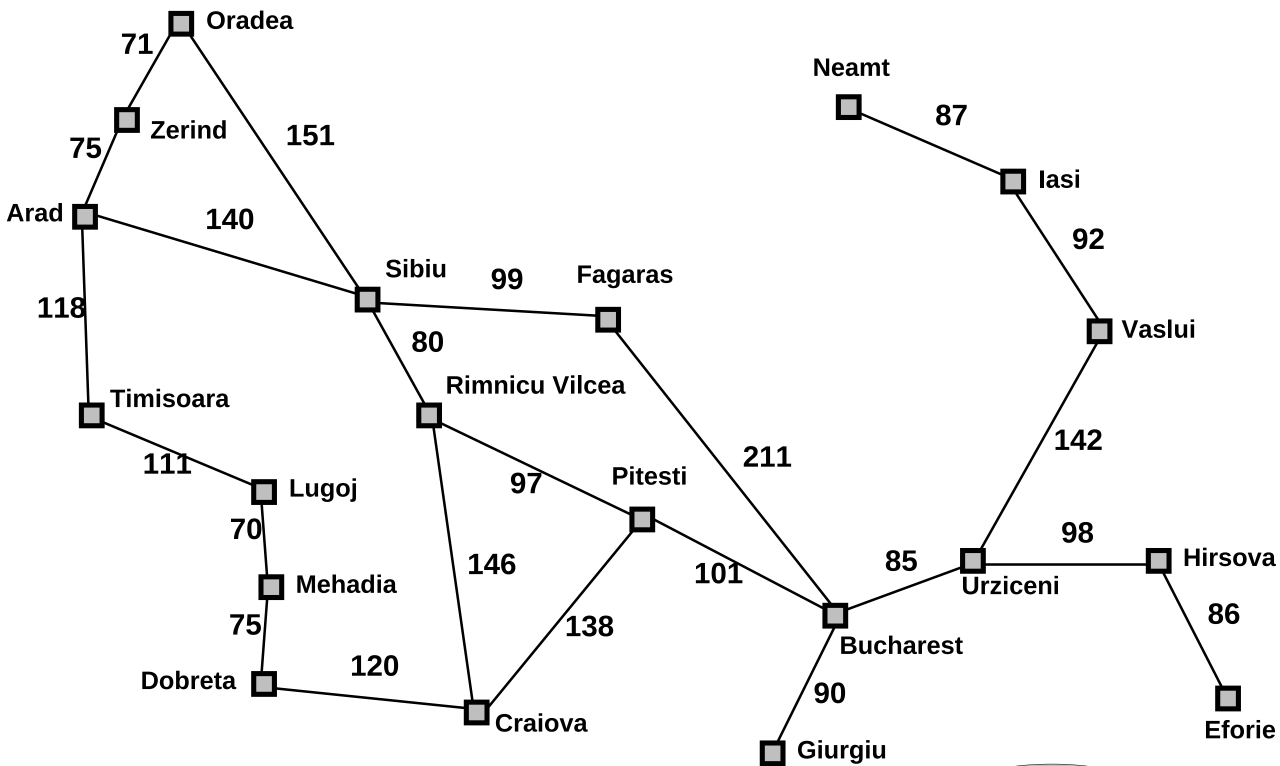




Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Search tree:

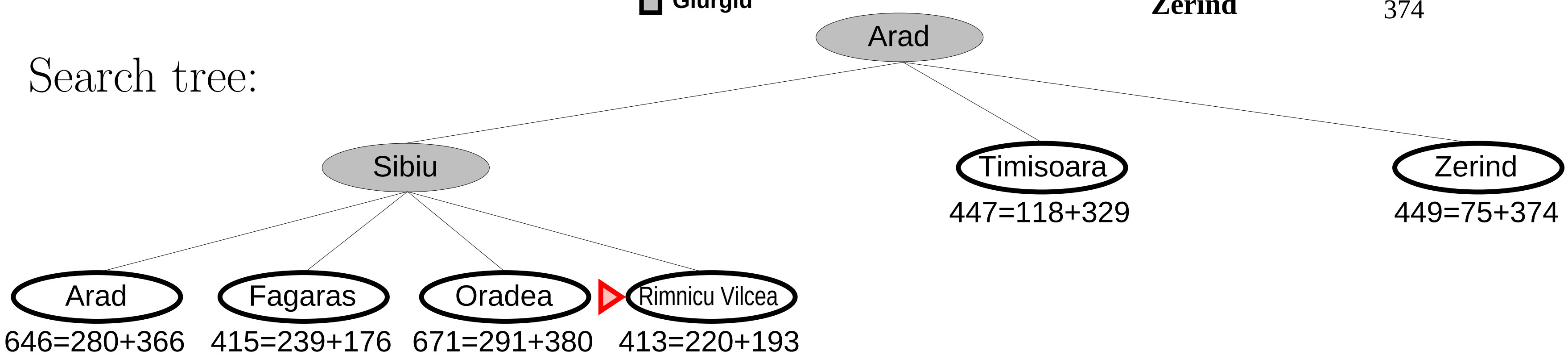


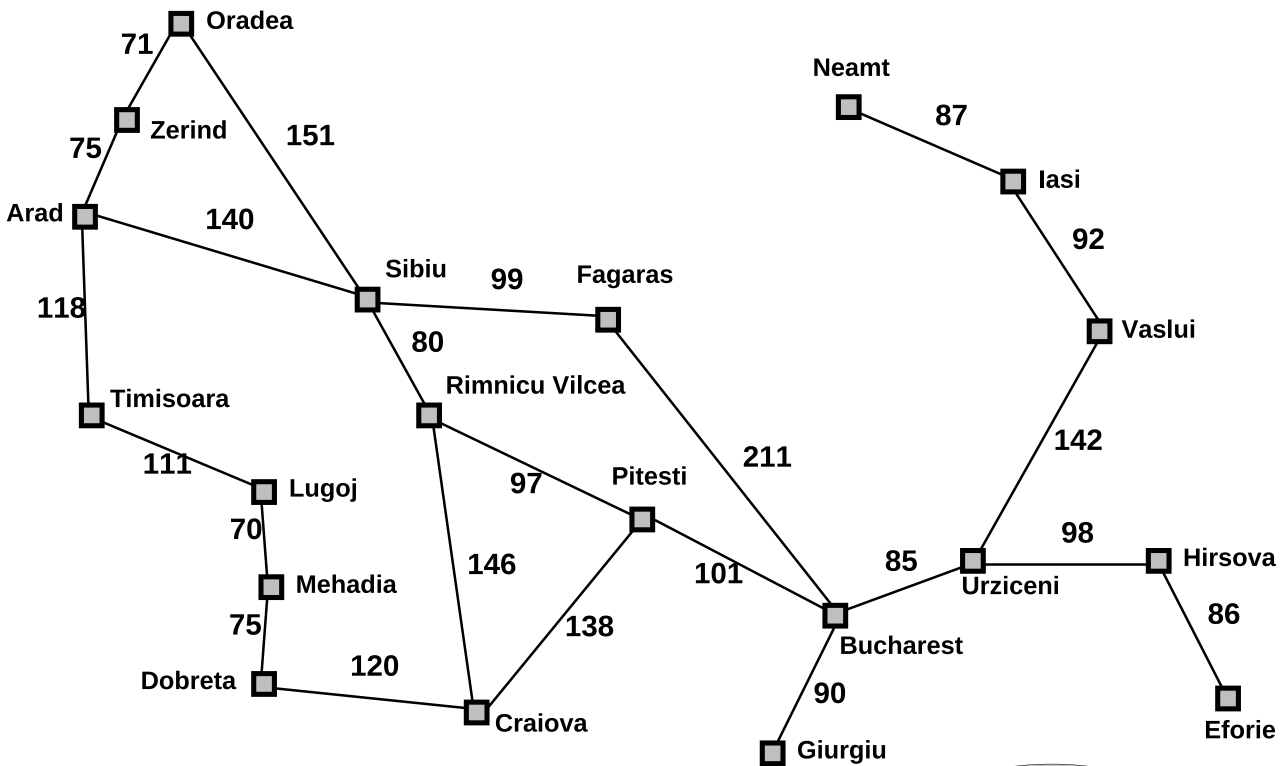


Straight−line distance to Bucharest

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

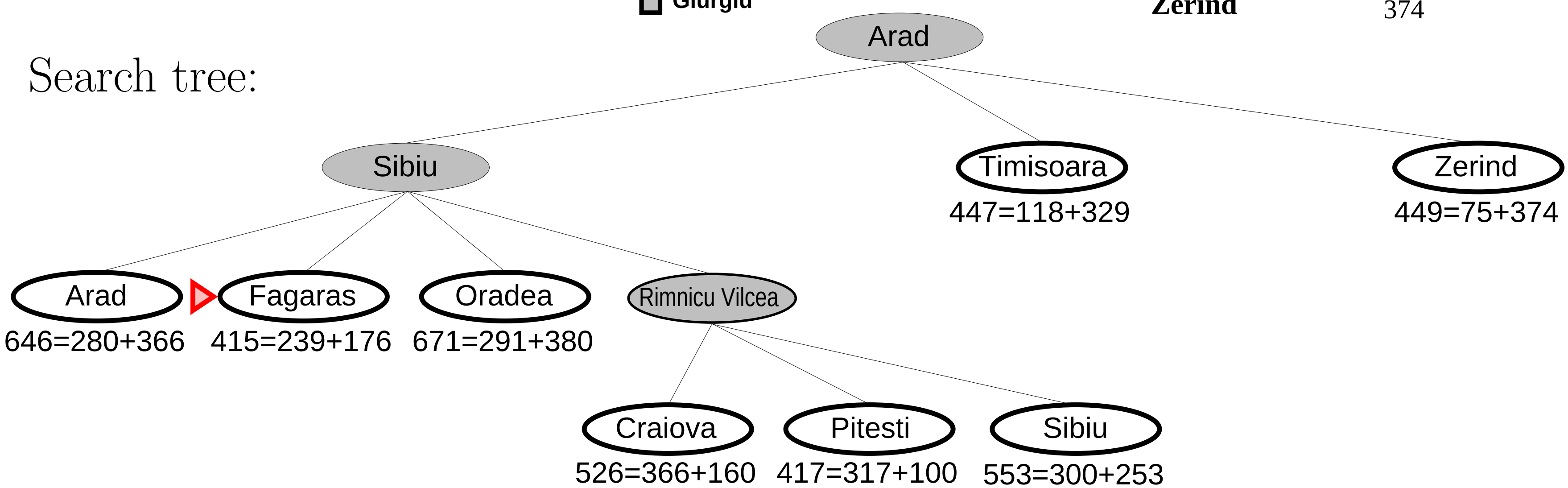
Search tree:

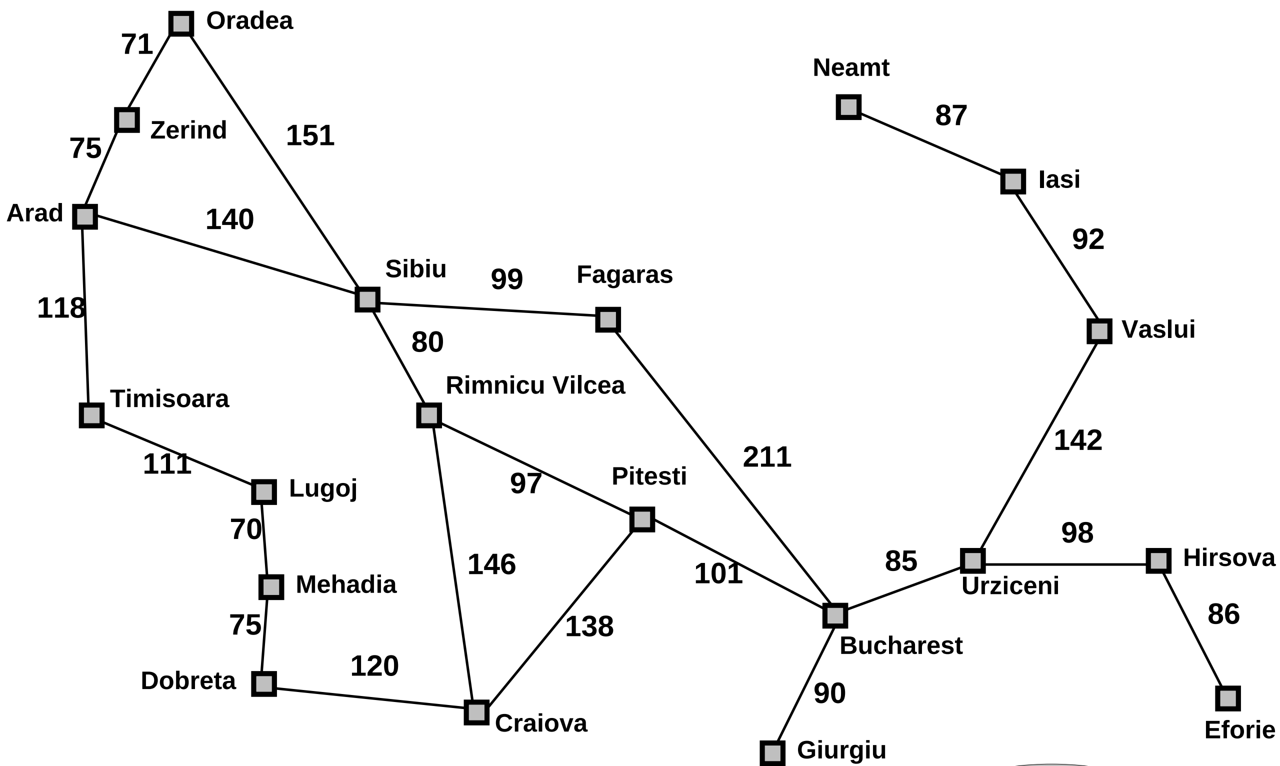




Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Search tree:

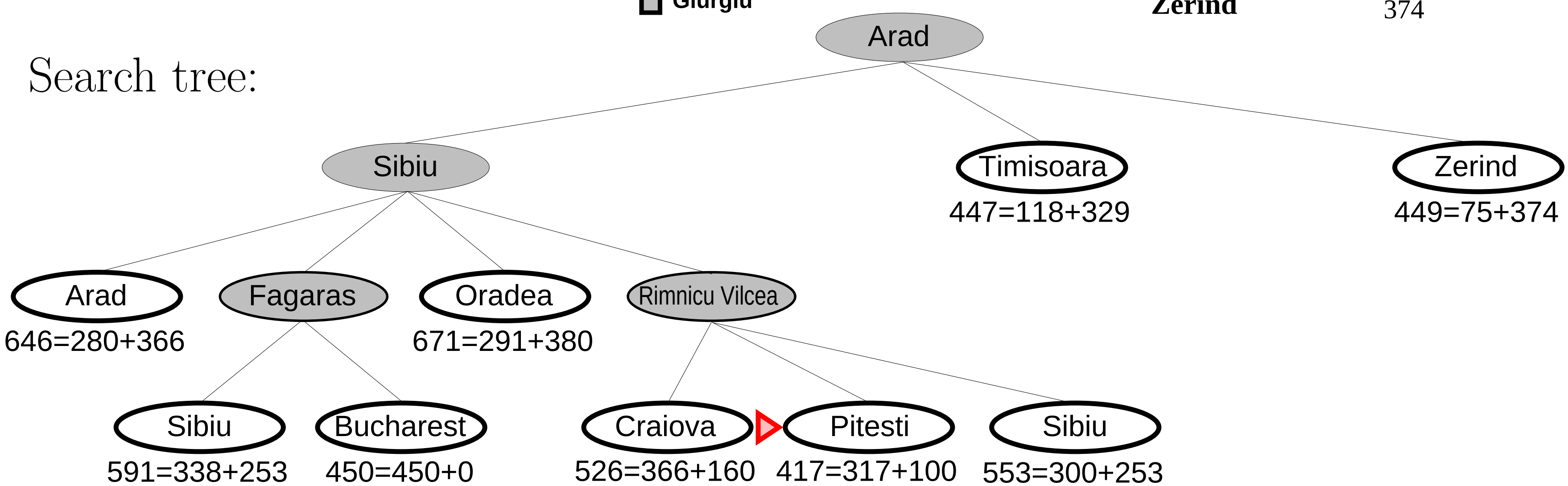


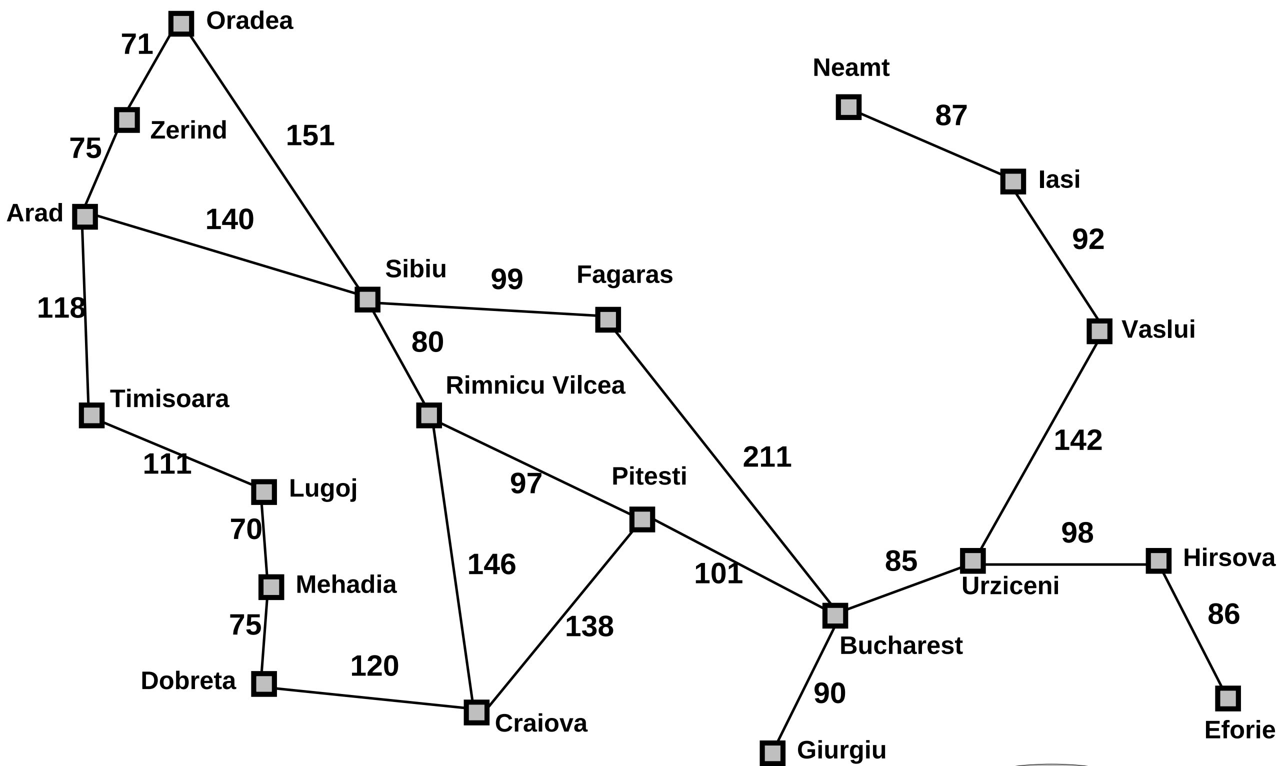


Straight−line distance to Bucharest

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

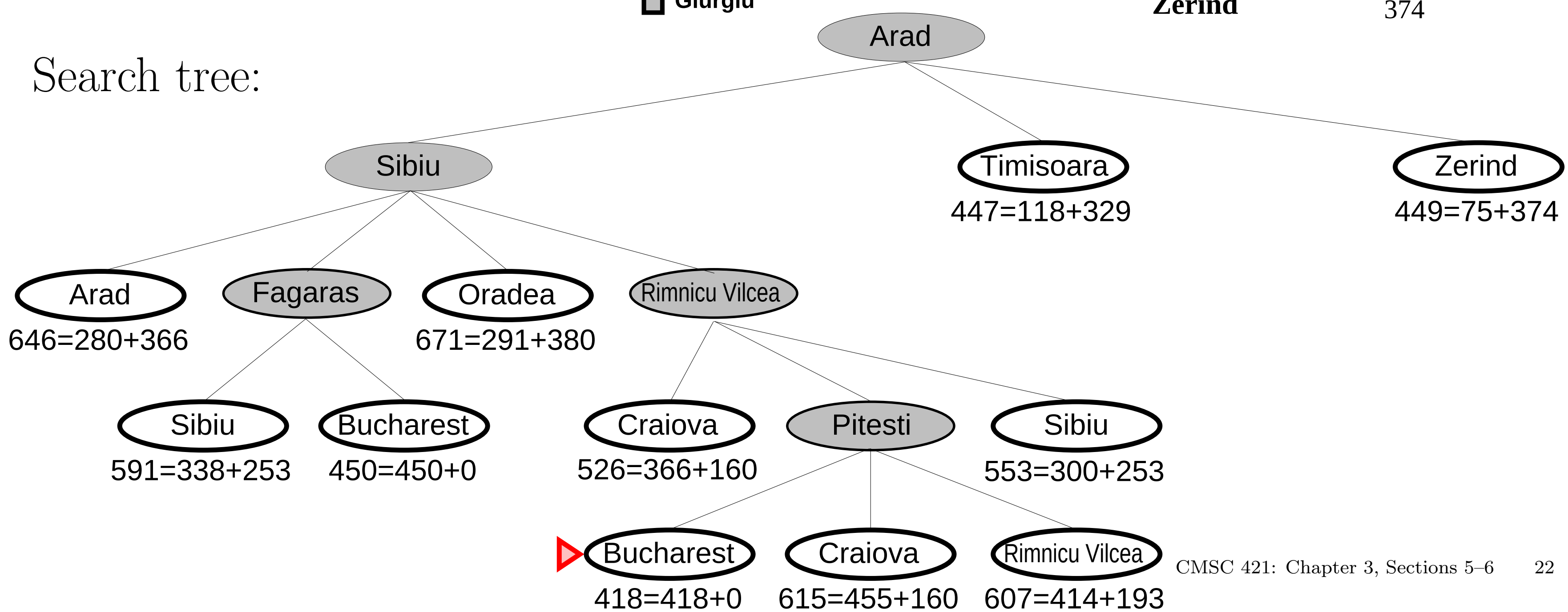
Search tree:





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

Search tree:



Admissibility

◇ Recall that

- $g(x)$ = cost to reach node x
- $h(x)$ = estimated cost from x to goal
- $f(x)$ = estimated total cost of path through x to goal

◇ h is *admissible* if it never overestimates the cost of getting to a goal

- $0 \leq h(x) \leq h^*(x)$ for every x,
  - ◇ where $h^*(x)$ = cost of best path from x to goal

◇ E.g., $h_{\mathrm{SLD}}(x)$ = straight-line distance from x to Bucharest

- This never overestimates the actual road distance to Bucharest

◇ If h is admissible then for every goal node x,

- $h(x) = 0$
- $f(x) = g(x)$

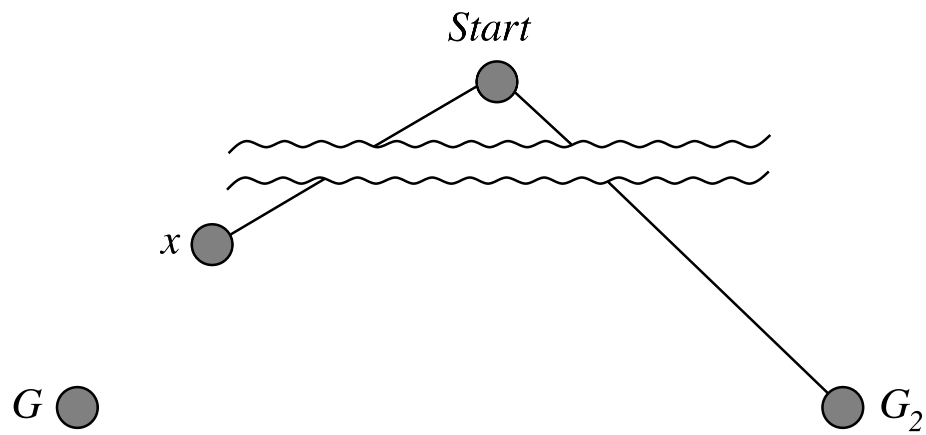
Optimality

Theorem: If h is admissible, then A* tree search will never return a non-optimal solution

◇ Let x be the first unexpanded node on an optimal path to an optimal goal node G

- Then x is in *frontier*

◇ Suppose *frontier* also contains a suboptimal goal node, G_2



$$
\begin{aligned}
f(G_2) &= g(G_2) + h(G_2) && \text{by definition} \\
&= g(G_2) && \text{since } G_2 \text{ is a goal} \\
&> g(G) && \text{since } G \text{ is optimal and } G_2 \text{ isn't} \\
&\geq g(x) + h(x) && \text{since } h \text{ is admissible} \\
&= f(x) && \text{by definition}
\end{aligned}
$$

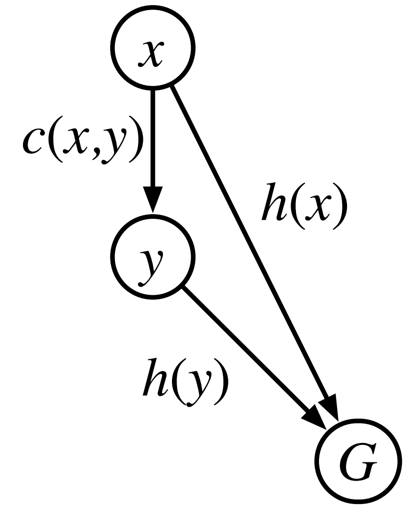
◇ A* won't select G_2, because *frontier* contains a node that looks better

Other properties

- *Completeness requirement* for A* tree search:
  - No infinite path has a finite cost
- **Theorem**: On any solvable problem that satisfies the completeness requirement, A* tree search will return a solution
- **Corollary**: If the admissibility requirement also is satisfied, then A* tree search will return an optimal solution

A* with Graph-Search

- A* can be used with Graph-Search, but there's a new requirement
- To guarantee optimality, the heuristic must be *consistent*
  - Analogous to the triangle inequality in geometry
  - For every node x and child y, $h(x) \leq c(x,y) + h(y)$



- If h is consistent, then

$$\begin{aligned} f(y) &= g(y) + h(y) \\ &= g(x) + c(x,y) + h(y) \\ &\geq g(x) + h(x) \\ &= f(x) \end{aligned}$$

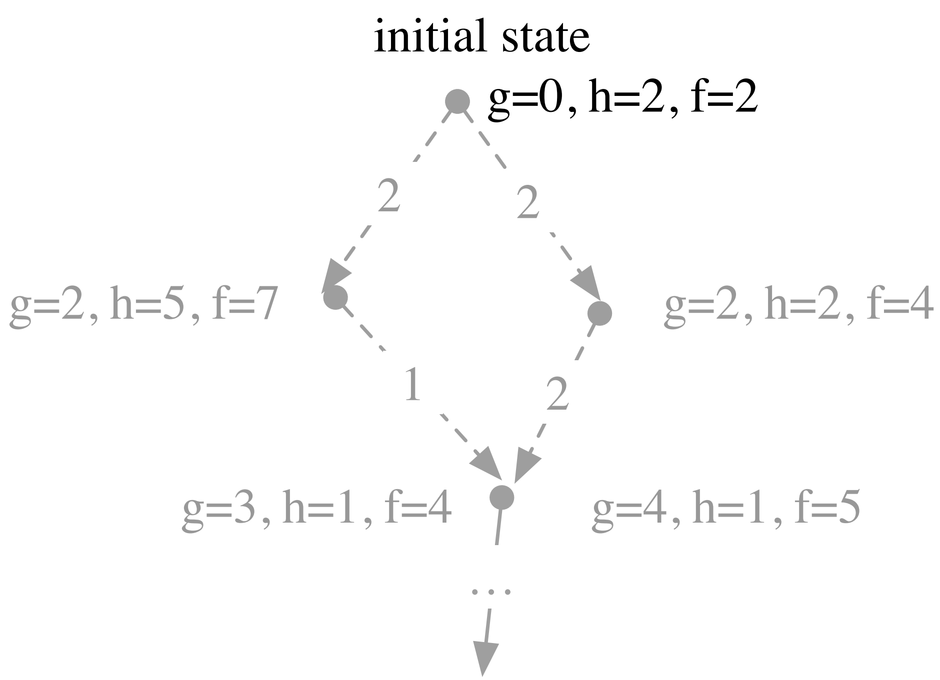
i.e., $f(x)$ is nondecreasing along any path.

- If h is consistent, then A* expands nodes in order of increasing f value

Behavior of A* with inconsistent heuristic

◇ If h is inconsistent, then

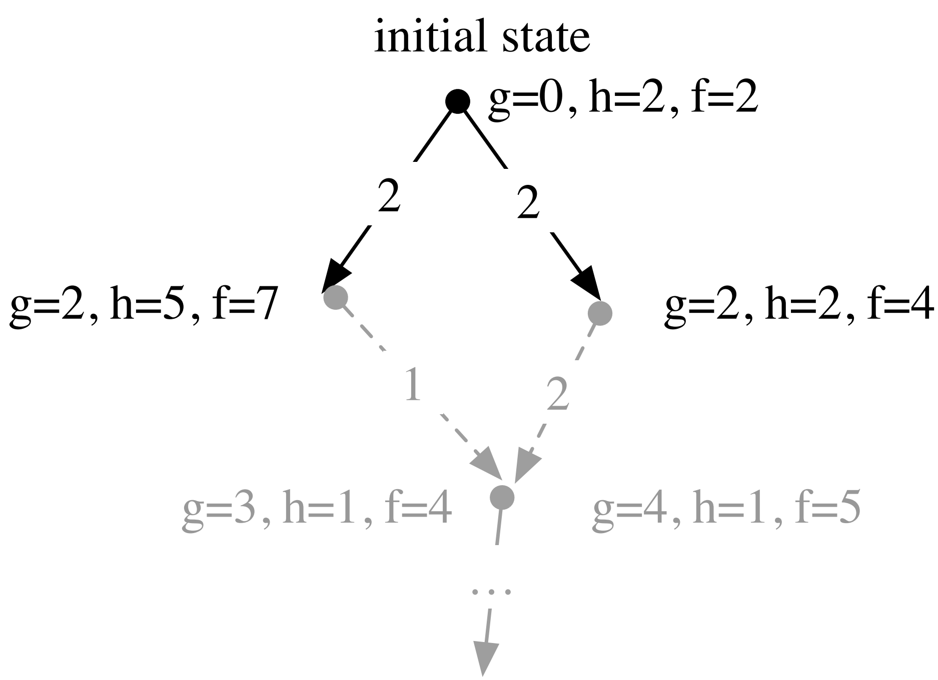
- As we go along a path, f may sometimes decrease
- A* may find lower-cost paths to states it has already expanded



Behavior of A* with inconsistent heuristic

◇ If h is inconsistent, then

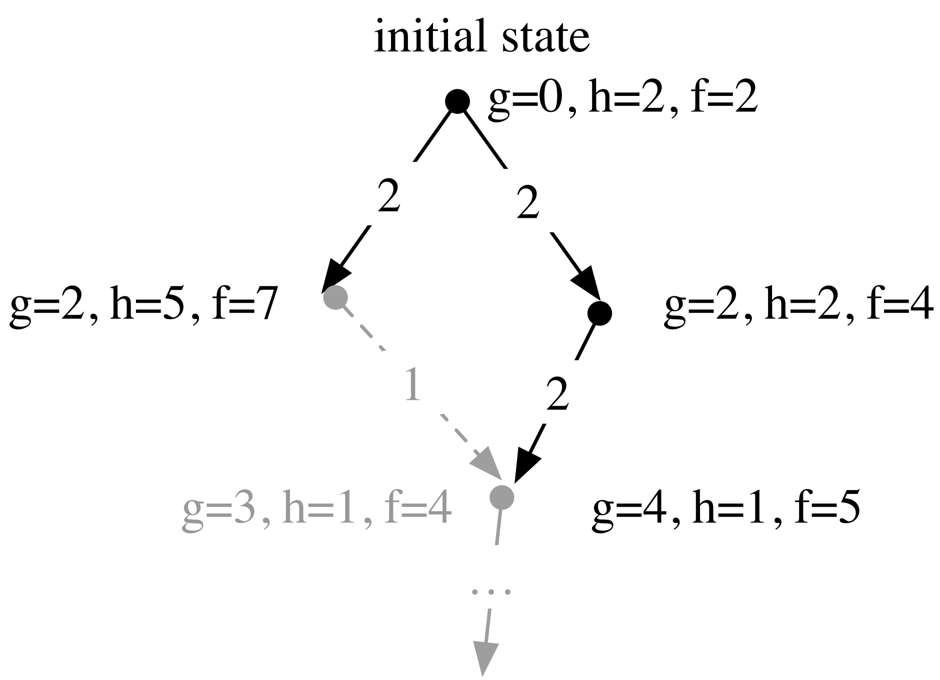
- As we go along a path, f may sometimes decrease
- A* may find lower-cost paths to states it has already expanded



Behavior of A* with inconsistent heuristic

◇ If h is inconsistent, then

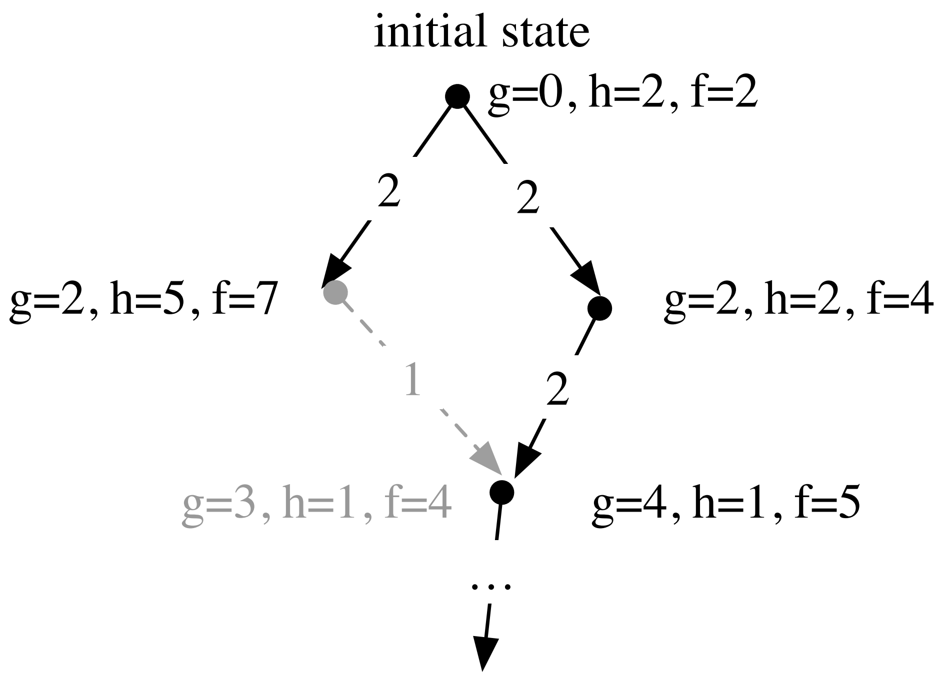
- As we go along a path, f may sometimes decrease
- A* may find lower-cost paths to states it has already expanded



Behavior of A* with inconsistent heuristic

♢ If h is inconsistent, then

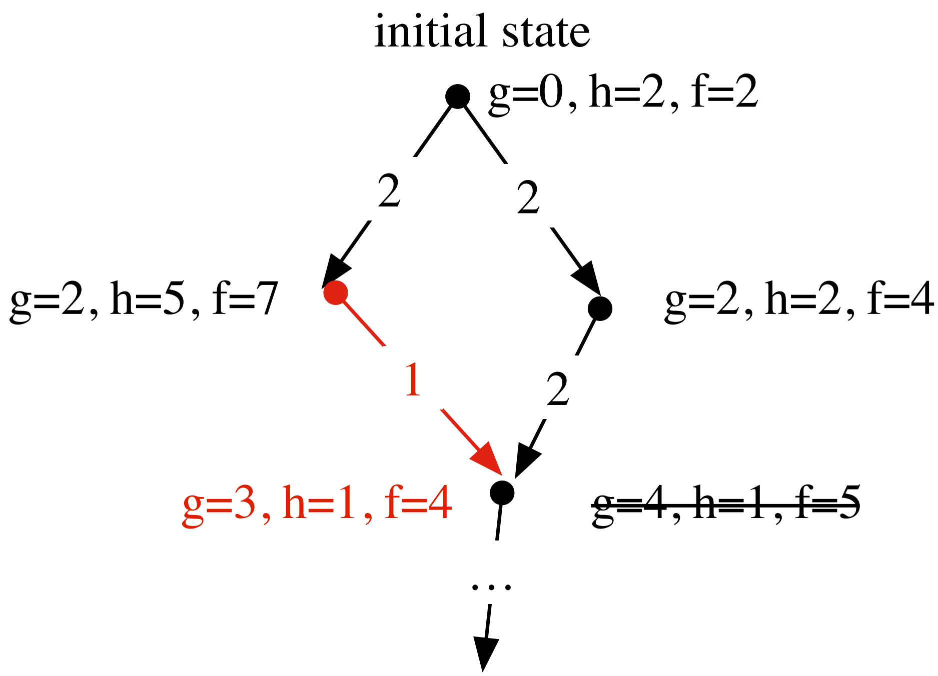
- As we go along a path, f may sometimes decrease
- A* may find lower-cost paths to states it has already expanded



Behavior of A* with inconsistent heuristic

◇ If h is inconsistent, then

- As we go along a path, f may sometimes decrease
- A* may find lower-cost paths to states it has already expanded



◇ Need to re-expand, but GRAPH-SEARCH won't do so

A* for graphs

◇ Find optimal solutions even if the heuristic is inconsistent

- Make *explored* a list of nodes, so we know their f values
- Re-expand states if better paths are found

function A*-GRAPH-SEARCH(*problem*)

explored ← an empty set
frontier ← list that contains a node for *problem*'s initial state
loop
  if *frontier* is empty **then return** Failure
  $x \leftarrow \arg\min\{f(y) \mid y \in \mathit{frontier}\}$
  if STATE[x] is a goal state **then return** x
  if there is no node y in *explored* such that
    STATE[y] = STATE[x] and $f(y) \leq f(x)$
  then
    add x to *explored*
    expand x and add the new nodes to *frontier*

Properties of A*

◊ *Complete?*

Properties of A*

- ◊ *Complete?* Yes, unless there are infinitely many nodes with $f \leq f(G)$
- ◊ *Time?*

Properties of A*

◇ <u>*Complete?*</u> Yes, unless there are infinitely many nodes with $f \leq f(G)$

◇ <u>*Time?*</u> O(entire state space) in worst case, $O(d)$ in best case

◇ <u>*Space?*</u>

Properties of A*

- <u>*Complete?*</u> Yes, unless there are infinitely many nodes with $f \leq f(G)$
- <u>*Time?*</u> O(entire state space) in worst case, $O(d)$ in best case
- <u>*Space?*</u> Keeps all nodes in memory
- <u>*Finds optimal solutions?*</u>

Properties of A*

- *Complete?* Yes, unless there are infinitely many nodes with $f \le f(G)$
- *Time?* O(entire state space) in worst case, $O(d)$ in best case
- *Space?* Keeps all nodes in memory
- *Finds optimal solutions?* Yes on trees
  - Also on graphs, if the heuristic is consistent
    or if we use the rewritten version of A*

- Additional properties:
  - A* expands all nodes in *frontier* that have $f(x) < C^*$
  - A* expands some nodes with $f(x) = C^*$
  - If f is *consistent*, A* expands no nodes with $f(x) > C^*$

How to create admissible heuristics

◇ Suppose P is a problem we're trying to solve

- Let $h^*(x)$ = minimum cost of any solution path from x

◇ Let Q be a **relaxation** of P

- Remove some constraints on what constitutes a solution

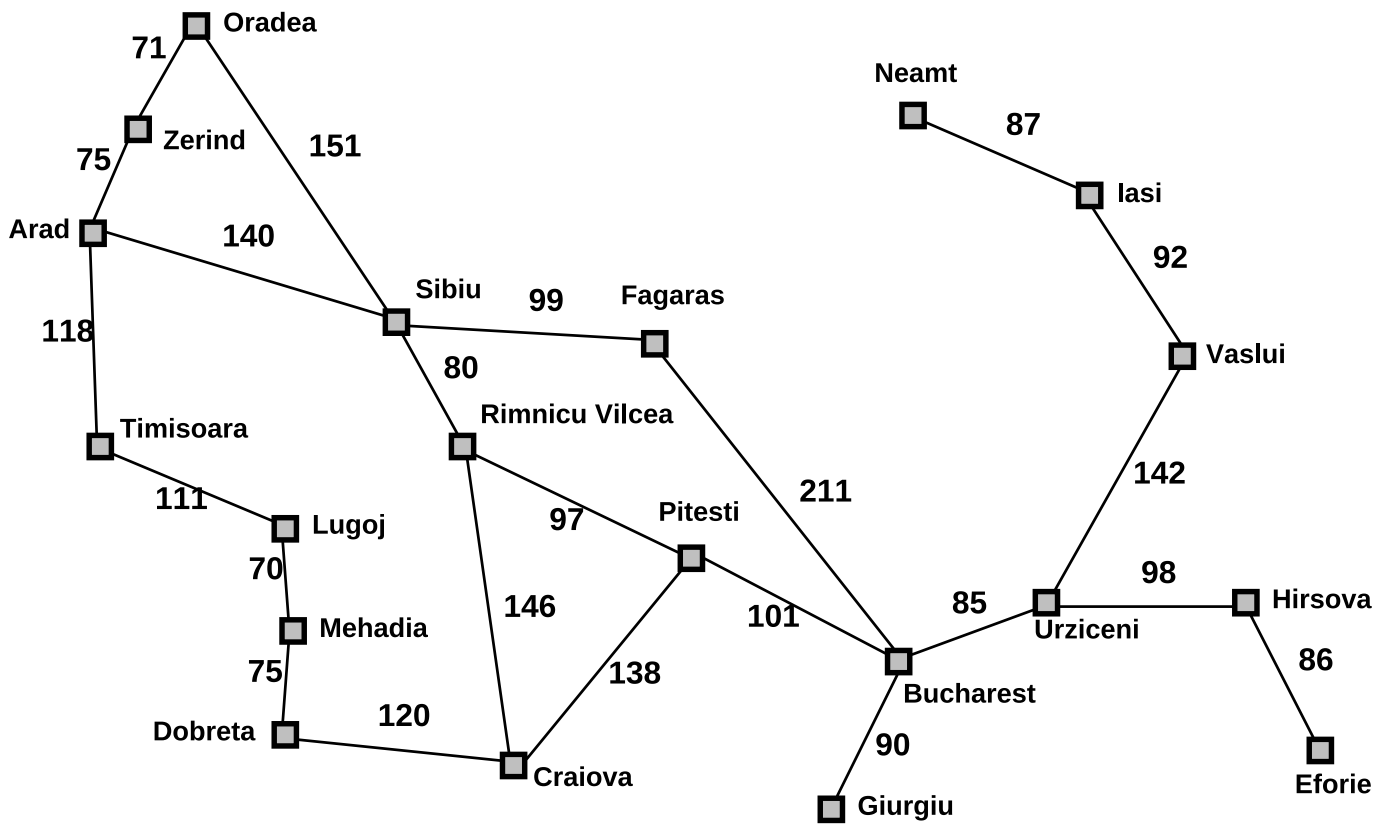
◇ Every solution path from x in P is also a solution path in Q

- Q may have some additional solution paths
- Some of them may cost less than $h^*(x)$

◇ Suppose that in Q, we can find optimal solutions quickly

- At x, find an optimal solution in Q
- Let $h(x)$ = cost of the solution we found
- Then $h(x) \leq h^*(x)$, i.e., h is an admissible heuristic for P

Example: Romania with step costs in km



Straight−line distance to Bucharest

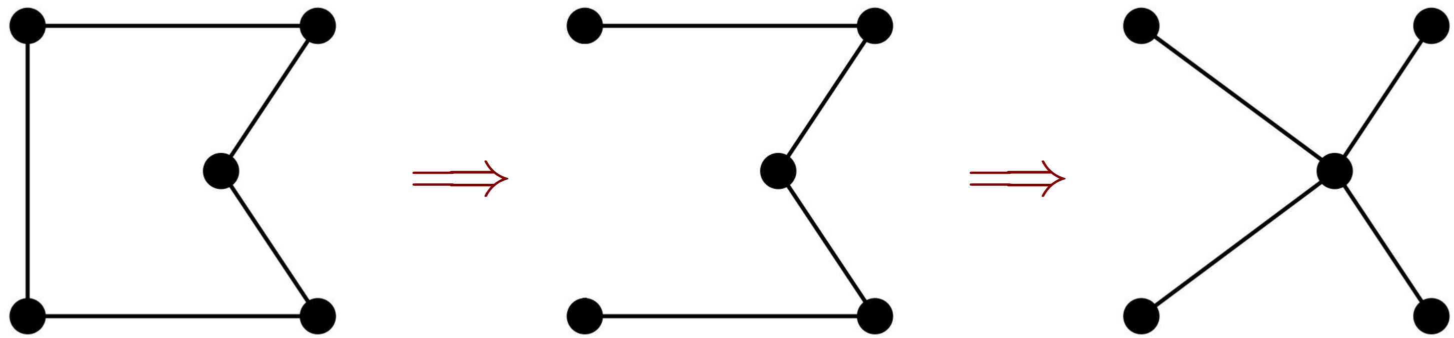
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

◇ Relax the problem to allow straight-line paths

- $h_{\text{SLD}}(x)$ cost of a straight line from city x to Bucharest

Example: TSP

- Well-known example: *traveling salesperson problem* (TSP)
  - Given a *complete* graph (edges between all pairs of nodes)
  - Find a least-cost *tour* (simple cycle that visits each city exactly once)



- Relax the problem twice:
  - Let {solutions} include acyclic paths that visit all cities
  - Let {solutions} include trees
- *Minimum spanning tree* can be computed in $O(n^2)$

  $\Rightarrow$ lower bound on the least-cost path that visits all cities

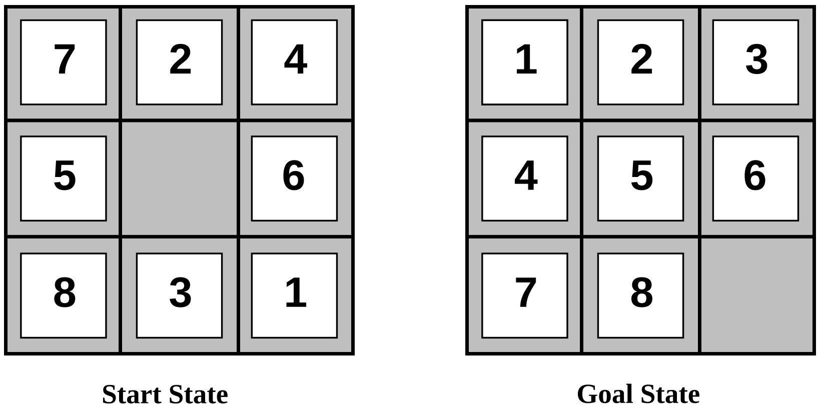
  $\Rightarrow$ lower bound on the least-cost tour

Example: the 8-puzzle

◇ Relaxation 1: allow a tile to move to any other square

- regardless of whether the square is adjacent
- regardless of whether there's another tile there already

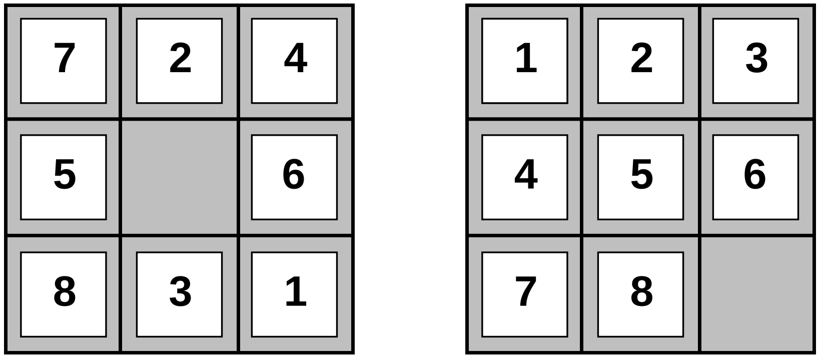
◇ This gives us $h_1(x)$ = number of misplaced tiles



$h_1(S)$ =?

Example: the 8-puzzle

- Relaxation 1: allow a tile to move to any other square
  - regardless of whether the square is adjacent
  - regardless of whether there's another tile there already
- This gives us $h_1(x)$ = number of misplaced tiles

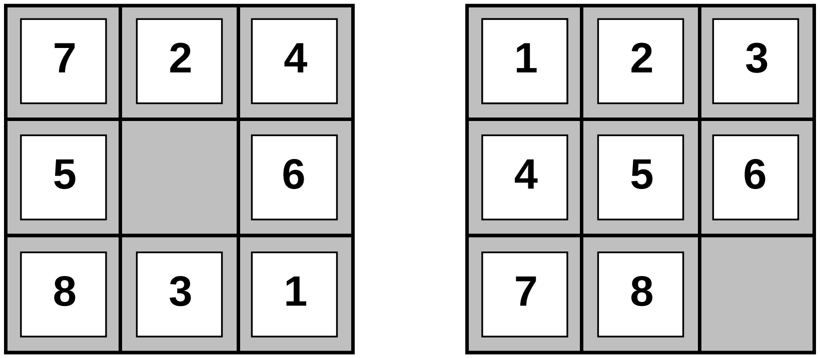


Start State Goal State

$h_1(S)$ =? 6

Example: the 8-puzzle

- ◇ Relaxation 2: allow a tile to move to any *adjacent* square
  - regardless of whether there's another tile there already
- ◇ This gives us $h_2(x)$ = total *Manhattan* distance

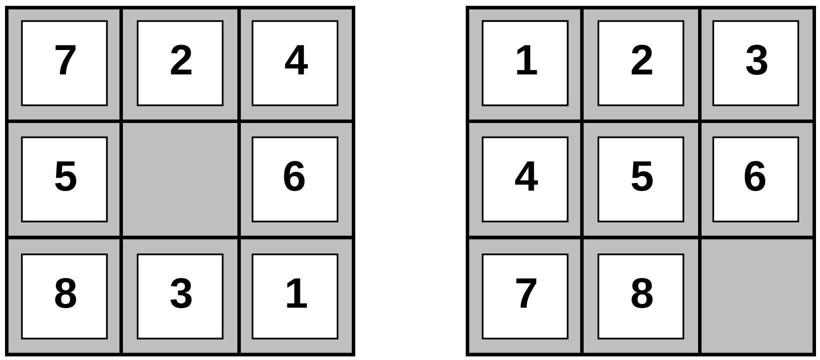


Start State **Goal State**

$h_2(S)$ =?

Example: the 8-puzzle

- ♢ Relaxation 2: allow a tile to move to any adjacent square
  - regardless of whether there's another tile there already
- ♢ This gives us $h_2(x)$ = total *Manhattan* distance



Start State **Goal State**

$h_2(S)$ =? $4+0+3+3+1+0+2+1 = 14$

Dominance

◇ $h_1(x)$ = number of misplaced tiles

◇ $h_2(x)$ = total *Manhattan* distance

◇ h_2 *dominates* h_1:

- $h_1(x) \leq h_2(x) \leq h^*(x)$ for all x,

◇ h_2's estimate of h^* is at least as good as h_1's, and often better

- Hence h_2 is better for search. Typical search costs:

$d = 14$:	IDS	3,473,941 nodes	
	A*(h_1)	539 nodes	
	A*(h_2)	113 nodes	
$d = 24$:	IDS	54,000,000,000 nodes	(approx.)
	A*(h_1)	39,135 nodes	
	A*(h_2)	1,641 nodes	

How to get dominance?

◇ If h_a are h_b admissible heuristic functions, then

- $h(x) = \max(h_a(x), h_b(x))$ is admissible and dominates both h_a and h_b

How to *fake* dominance:

◇ $h_2(x) = ch_1(x)$, where c is a positive constant

◇ Probably h_2 won't be admissible

$\Longrightarrow$ Won't always get optimal solutions

- But you may be able to get solutions much faster

◇ The larger you make c, the closer you get to a greedy search

Iterative-Deepening A*

function IDA*(*problem*) **returns** a solution
    inputs: *problem*, a problem

    $fmax \leftarrow h(initial\ state)$
    for $i \leftarrow 0$ **to** ∞ **do**
        $result \leftarrow$ LIMITED-F-SEARCH(*problem, fmax*)
        if *result* is a solution **then return** *result*
        else $fmax \leftarrow$ result
    end

function LIMITED-F-SEARCH(*problem, fmax*) **returns** solution or number

    depth-first search, backtracking at every node n such that $f(n) > fmax$
    if the search finds a solution
    then return the solution
    else return $\min\{f(m) \mid \text{the search backtracked at node } m\}$

Properties of IDA*

◊ *Complete?*

Properties of IDA*

◊ *Complete?* Yes, unless there are infinitely many nodes with $f(x) \leq f(G)$

◊ *Time?*

Properties of IDA*

- *Complete?* Yes, unless there are infinitely many nodes with $f(x) \leq f(G)$
- *Time?* Like A* **if** $f(x)$ is an integer and
  the number of nodes with $f(x) \leq k$ grows exponentially with k
- *Space?*

Properties of IDA*

- *Complete?* Yes, unless there are infinitely many nodes with $f(x) \leq f(G)$
- *Time?* Like A* **if** $f(x)$ is an integer and
  the number of nodes with $f(x) \leq k$ grows exponentially with k
- *Space?* $O(bd)$
- *Optimal?*

Properties of IDA*

◊ *Complete?* Yes, unless there are infinitely many nodes with $f(x) \leq f(G)$

◊ *Time?* Like A* **if** $f(x)$ is an integer and
the number of nodes with $f(x) \leq k$ grows exponentially with k

◊ *Space?* $O(bd)$

◊ *Optimal?* Yes

◊ With consistent heuristic:

- IDA* cannot expand f_{i+1} until f_i is finished
- IDA* expands all nodes with $f(x) < C^*$
- IDA* expands no nodes with $f(x) \geq C^*$

Summary

◇ Heuristic functions estimate costs of shortest paths

◇ Good heuristics can dramatically reduce search cost

◇ Greedy search expands lowest h

- incomplete, solutions not always optimal

◇ A* search expands lowest $g + h$

- Completeness and optimality if some requirements are satisfied
  - ◇ admissibility, etc.
- Can get admissible heuristics by solving relaxed problems

◇ IDA* is like a combination of A* and IDS

- much lower space requirement than A*
- same big-O time *if* number of nodes grows exponentially with cost