南京大学 人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

Lecture 3: Search 2

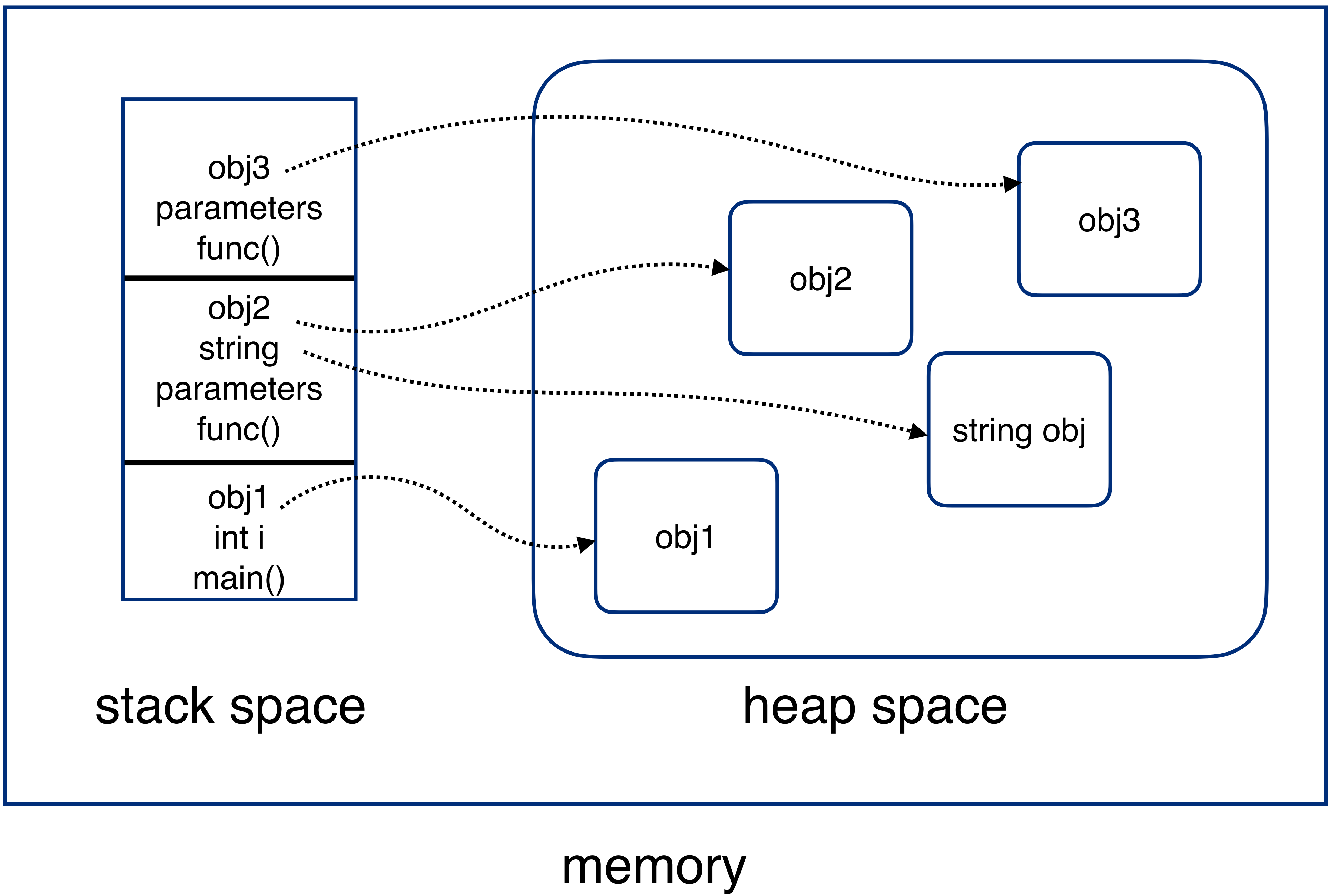
Previously...

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

note the time of goal-test: expanding time not generating time

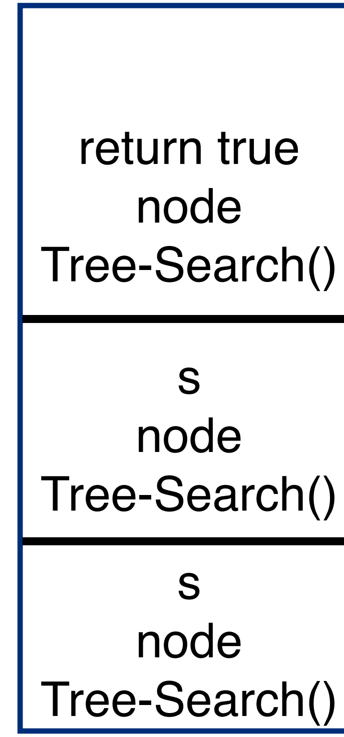
```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Stack and heap memory space



Deep-first search using stack

```
function Tree-Search(node)
    if node has goal then return true
    for each action, result in Successor-Fn(problem, node) do
        s <- make Node from node
        hasgoal = Tree-Search(s)
        if hasgoal then return true
    end for
return false
```



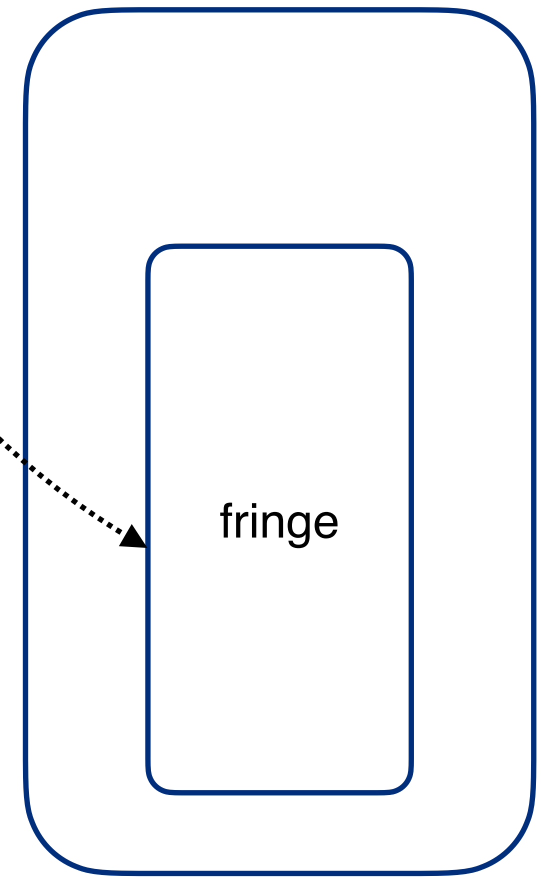
stack space

simple to code, risk of stack-overflow

Deep-first search using heap

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;   ACTION[s] ← action;   STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```



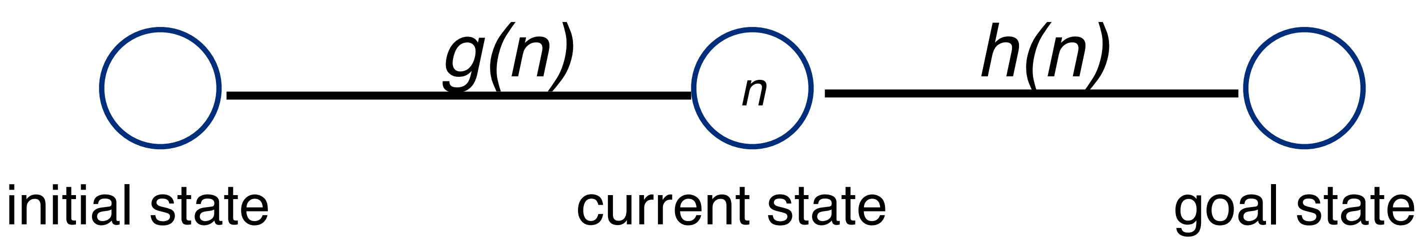
heap space

flexible memory usage

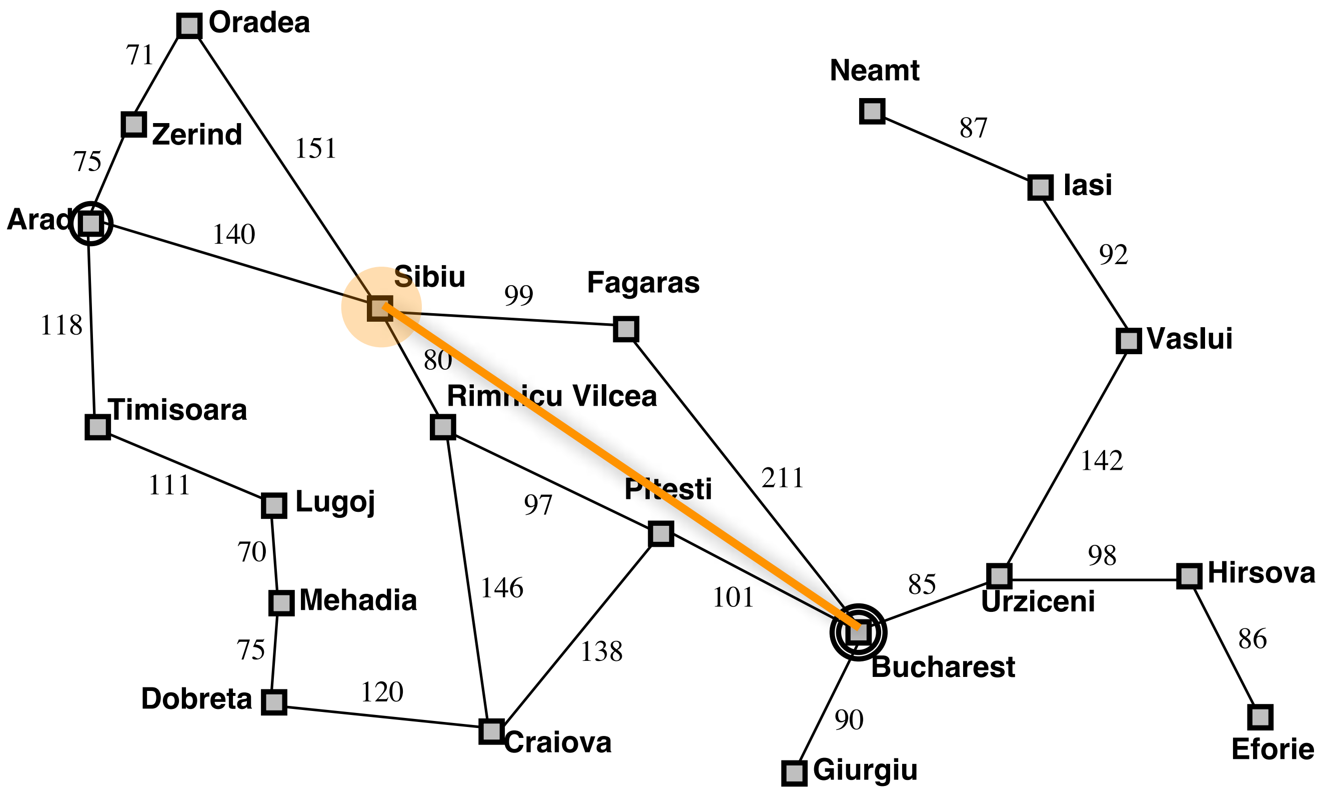
Informed Search Strategies

best-first search: *f*　　　　but what is best?

uniform cost search: cost function *g*
heuristic function: *h*



Example: h_{SLD}



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

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

Greedy search

Evaluation function $h(n)$ (**h**euristic)

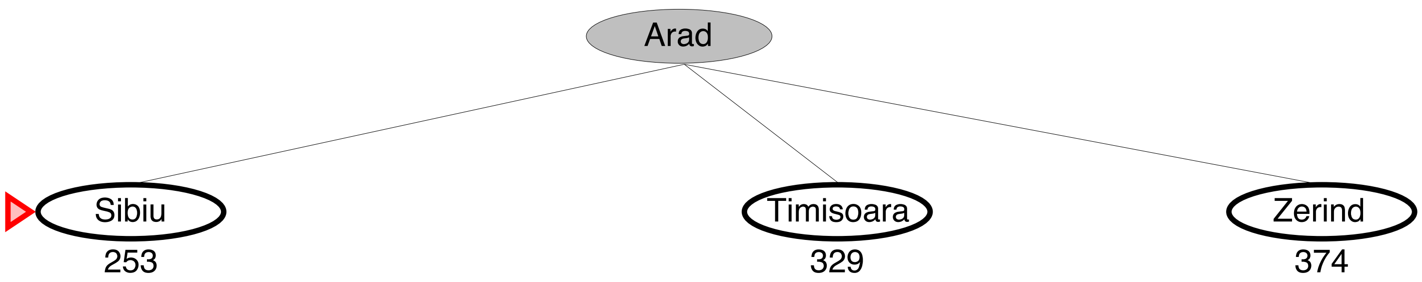
$= $ estimate of cost from n to the closest goal

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

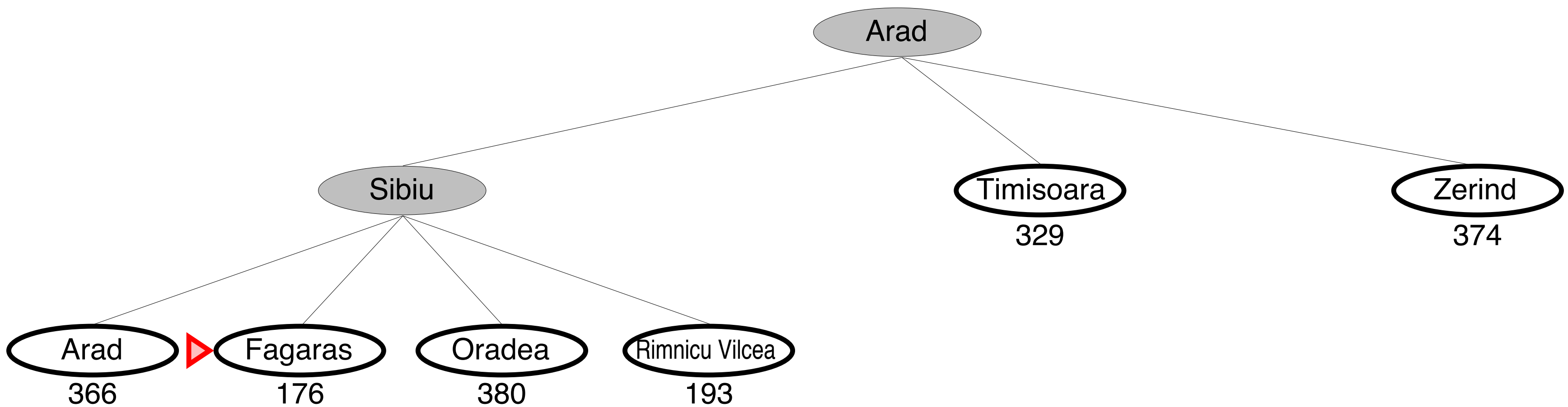
Greedy search expands the node that **appears** to be closest to goal

Example

Example

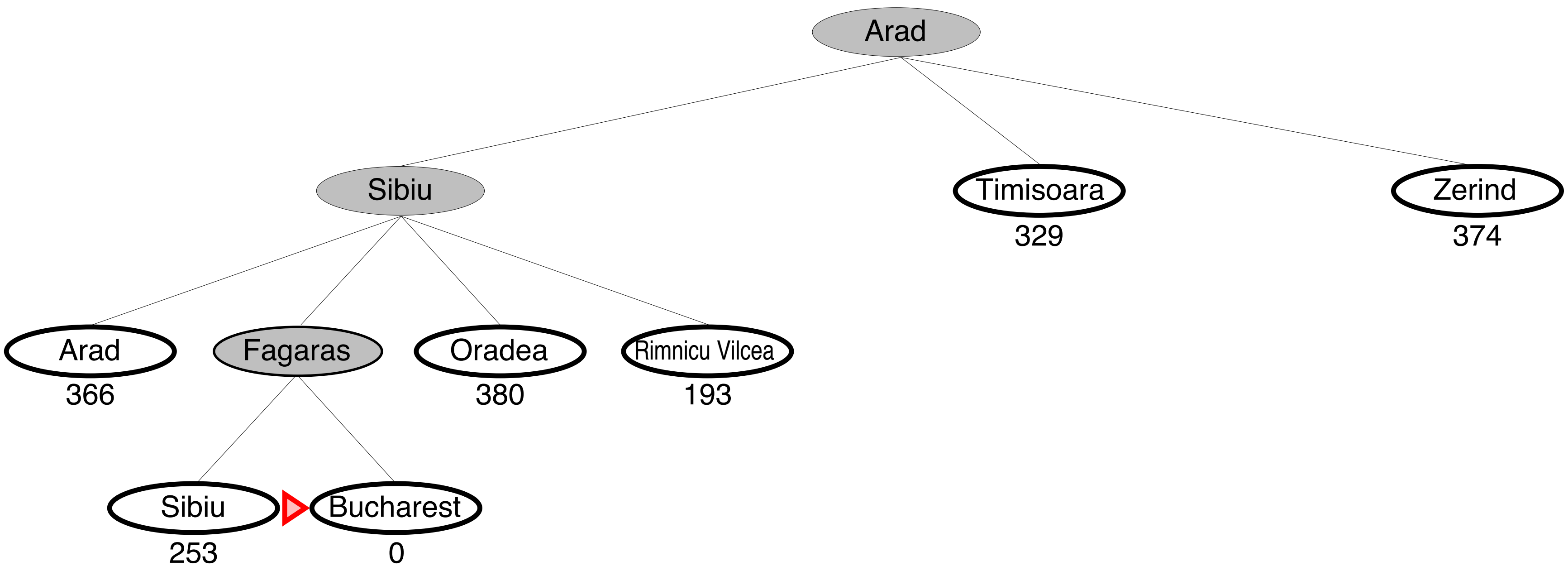


Example



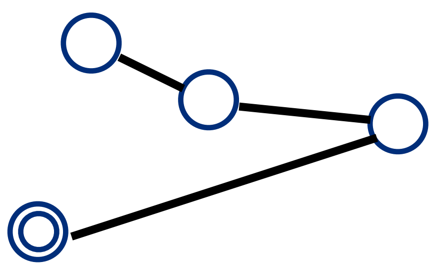
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193

Example


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

Properties

Complete?? No–can get stuck in loops, e.g.,



Complete in finite space with repeated-state checking

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

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

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

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

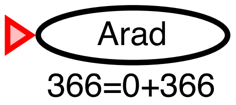
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

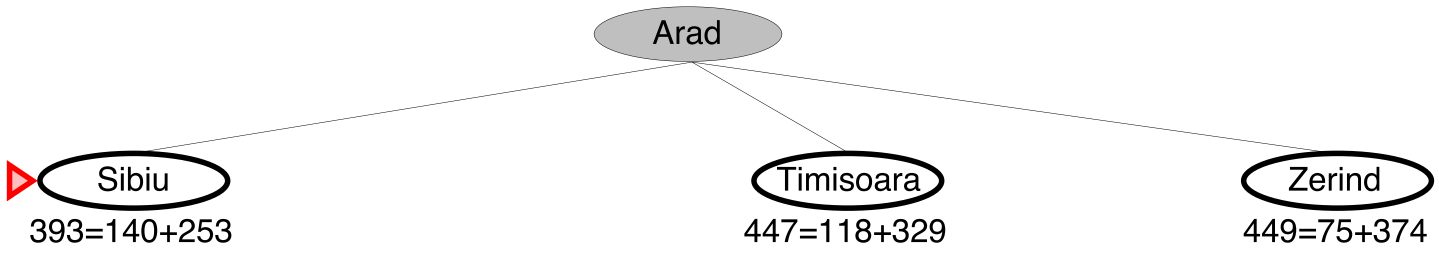
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

Example

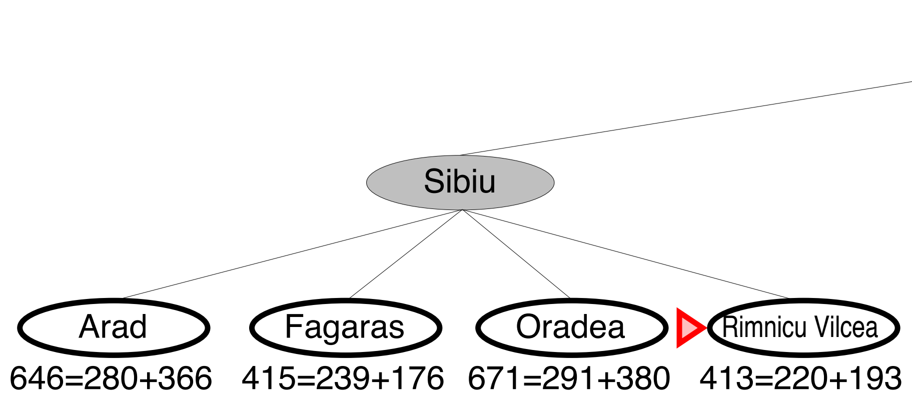


Example



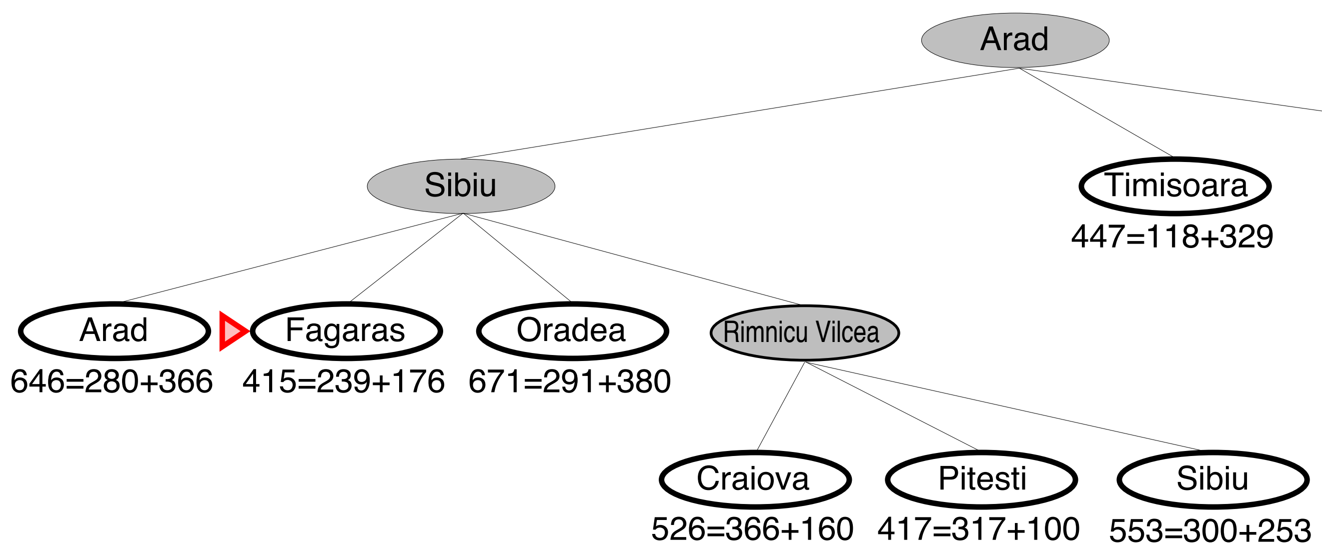
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

Example

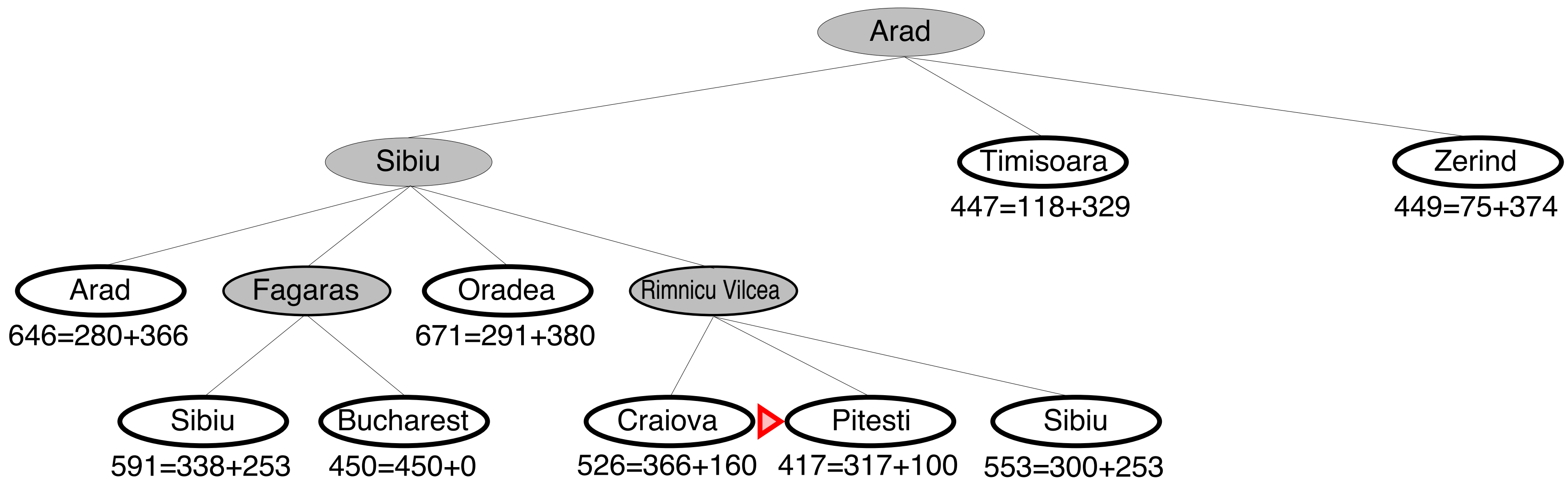


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

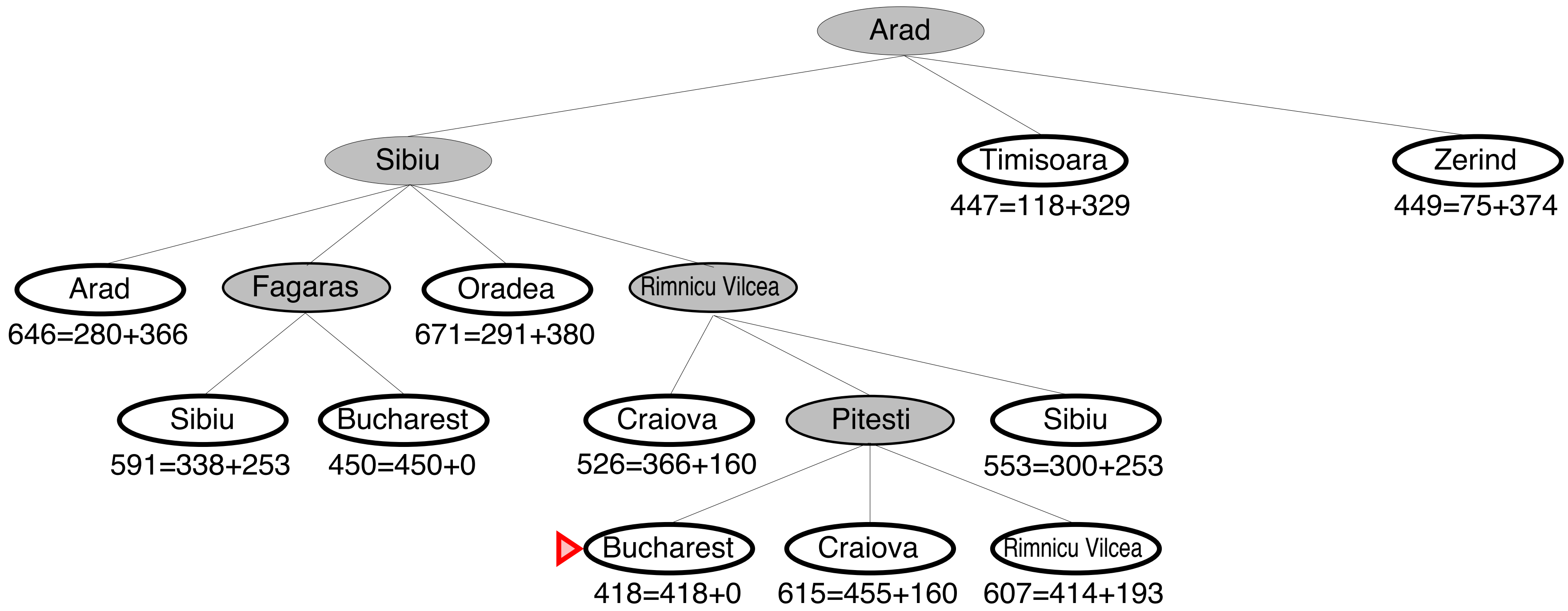
Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

Example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

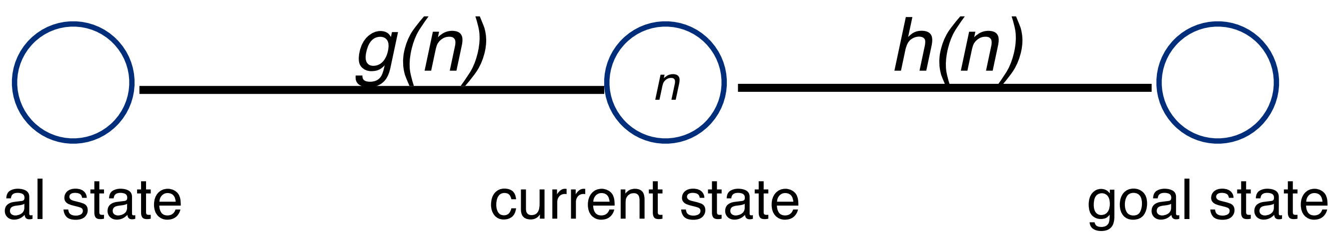
Example



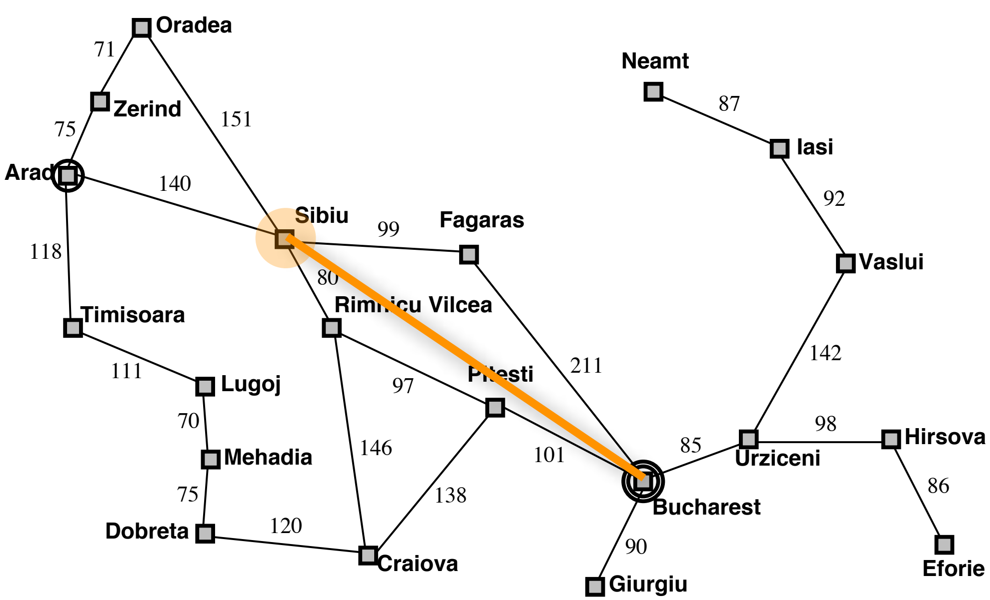
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* is optimal: Admissible and consistency

Admissible: never over estimate the cost



no larger than the cost of the optimal path from *n* to the goal



A* is optimal: Admissible and consistency

A* is optimal with admissible heuristic 重点理解!

why?

A* is optimal: Admissible and consistency

A* is optimal with admissible heuristic 重点理解!

why? 1. when a search algorithm is optimal?

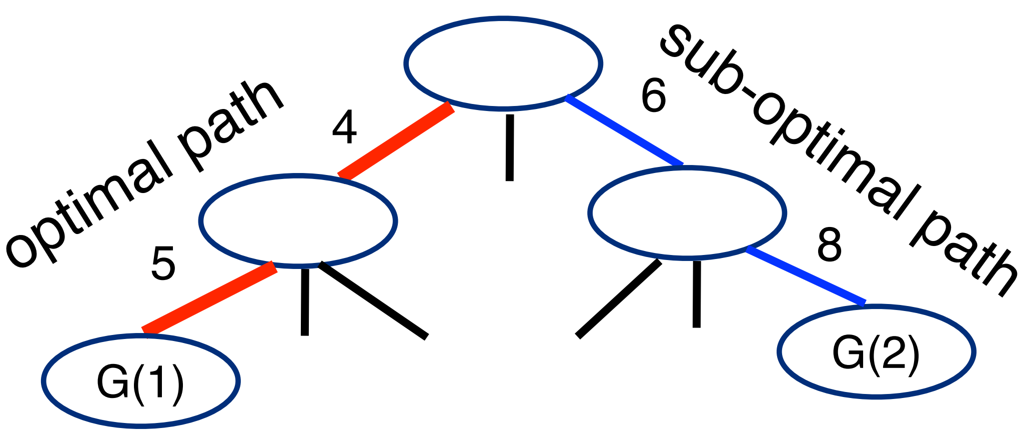
uniform cost search is optimal, because

a) it expands node with the smallest cost

b) the goal state on the optimal path has smaller cost than that on any sub-optimal path

c) it will never expand the goal states on sub-optimal paths before the goal state on the optimal path

key, the goal state on the optimal path has a smaller value than that on any sub-optimal paths



A* is optimal: Admissible and consistency

A* is optimal with admissible heuristic 重点理解!

why? 2. when the f=g+h value of the goal state on the optimal path is smaller than that on any sub-optimal path?

A* is optimal: Admissible and consistency

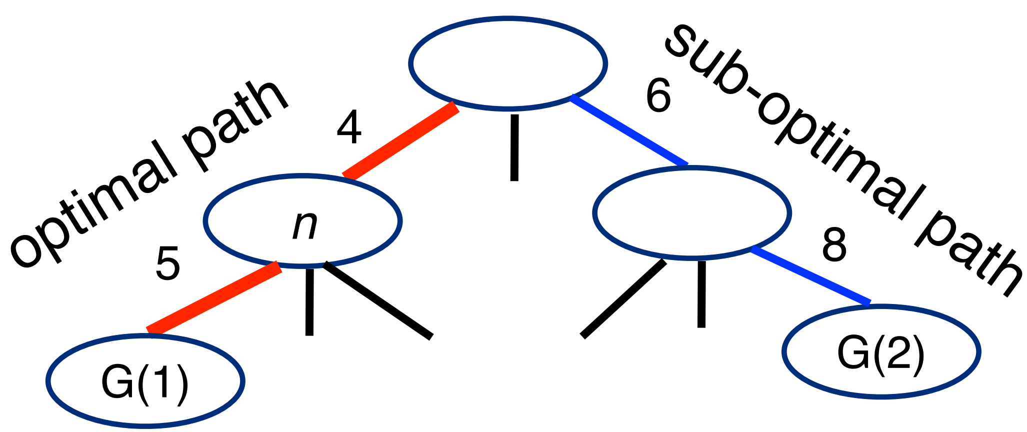
A* is optimal with admissible heuristic 重点理解!

why? 3. if h(n) <= h*(n), that is, the heuristic value is smaller than the true cost

for any node *n* on the optimal path

$$f(n) = g(n) + h(n) <= g(n) + h^*(n) = g(G(1)) <= g(G(2))$$

so n is always expanded before the goal state on any other sub-optimal path



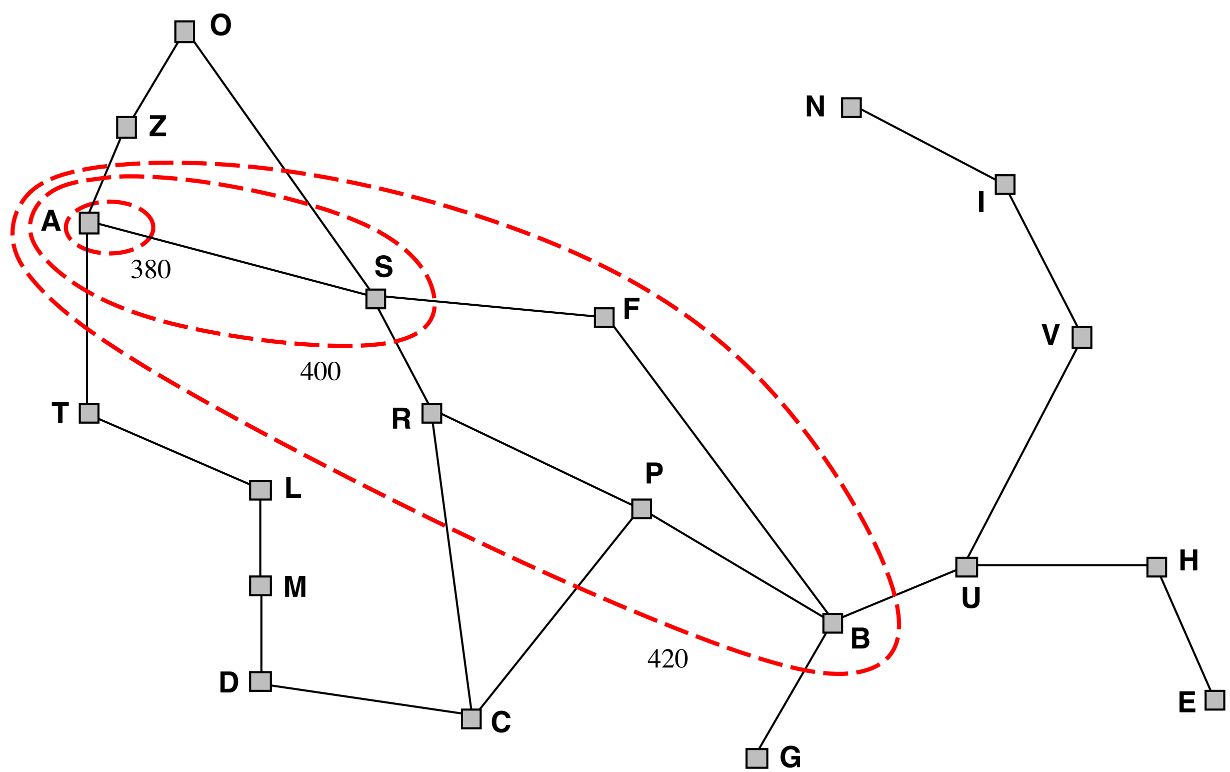
A* is optimal: Admissible and consistency

A* is optimal with admissible heuristic

why?

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



A* is optimal: Admissible and consistency

Admissible is for tree search, for graph search

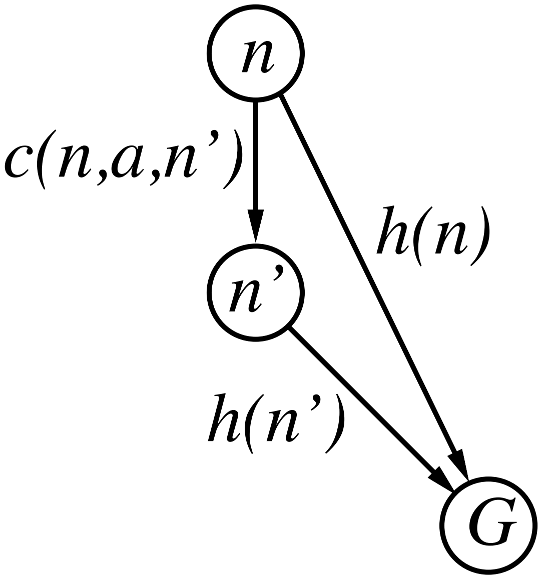
A heuristic is consistent if

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

If h is consistent, we have

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

I.e., $f(n)$ is nondecreasing along any path.

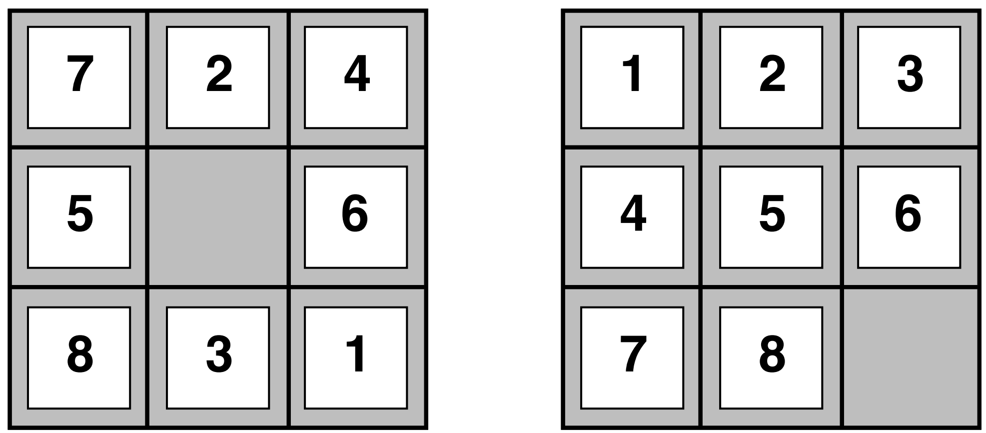


Proof is similar with that of admissible

Example

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

$h_1(S) =$?? 6
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

Dominance

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

why?

Typical search costs:

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

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Admissible heuristics from relaxed problem

Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem

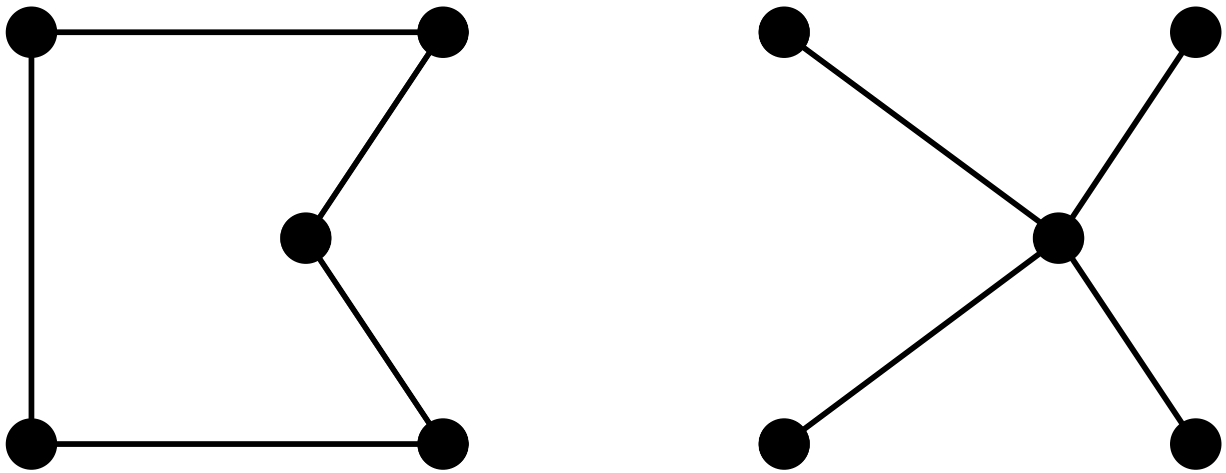
If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ gives the shortest solution

If the rules are relaxed so that a tile can move to **any adjacent square**, then $h_2(n)$ gives the shortest solution

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Example

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once



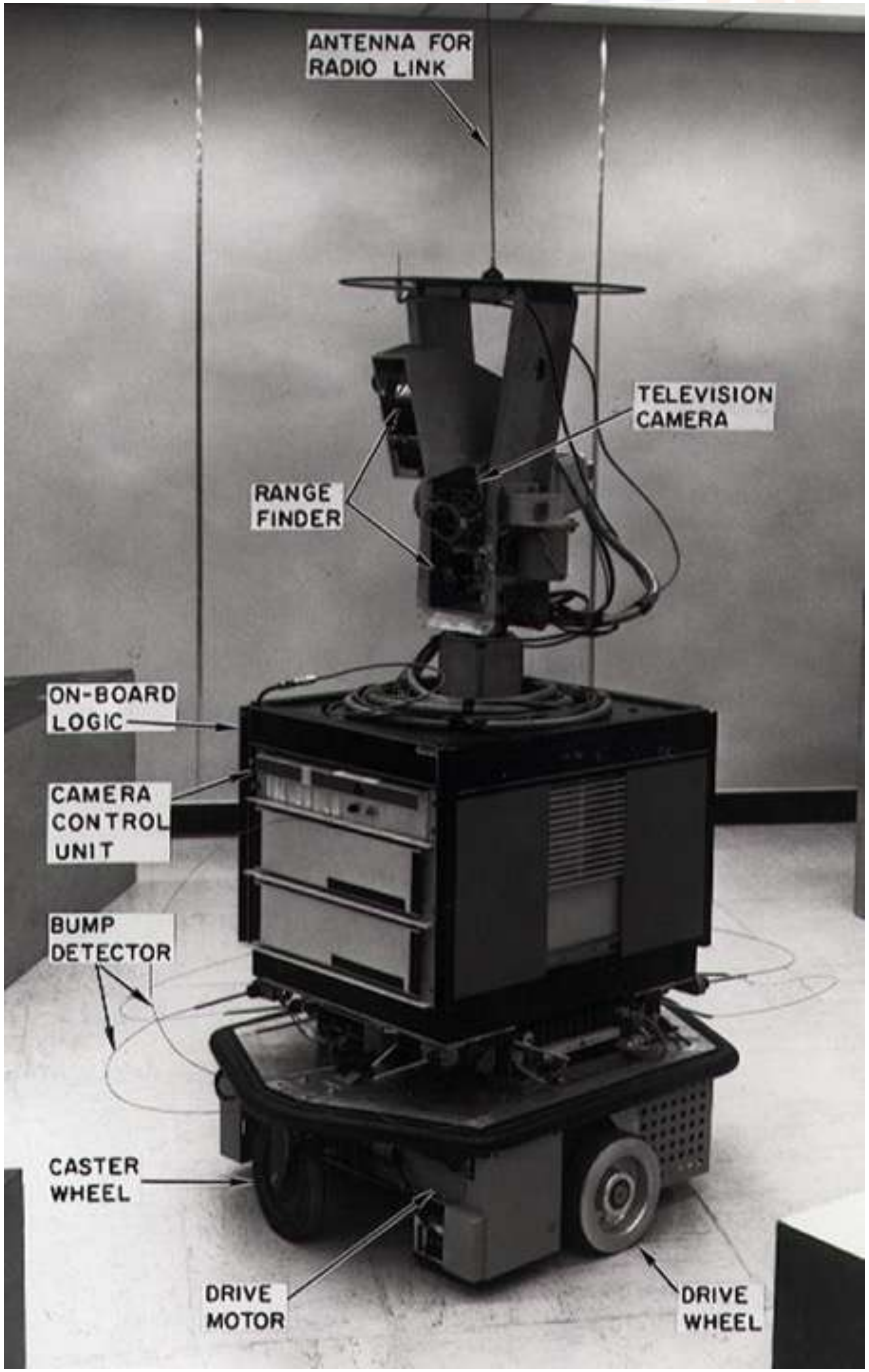
Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

Where did A* come from

Shakey 50 Years

Shakey the robot was the first general-purpose mobile robot to be able to reason about its own actions

Developed in SRI International from 1966



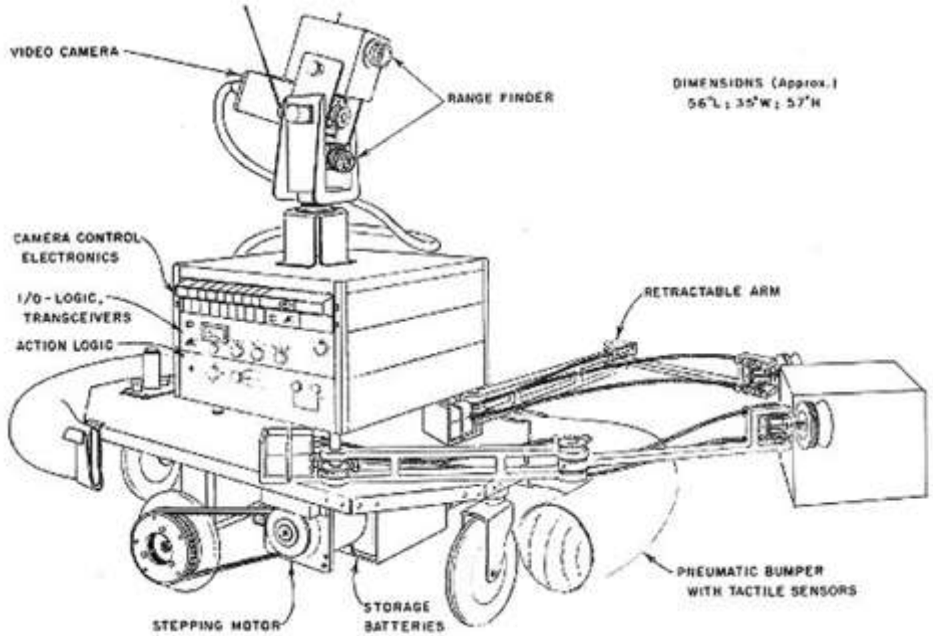


FIG. 2 AUTOMATON VEHICLE

Celebration of Shakey in AAAI'15