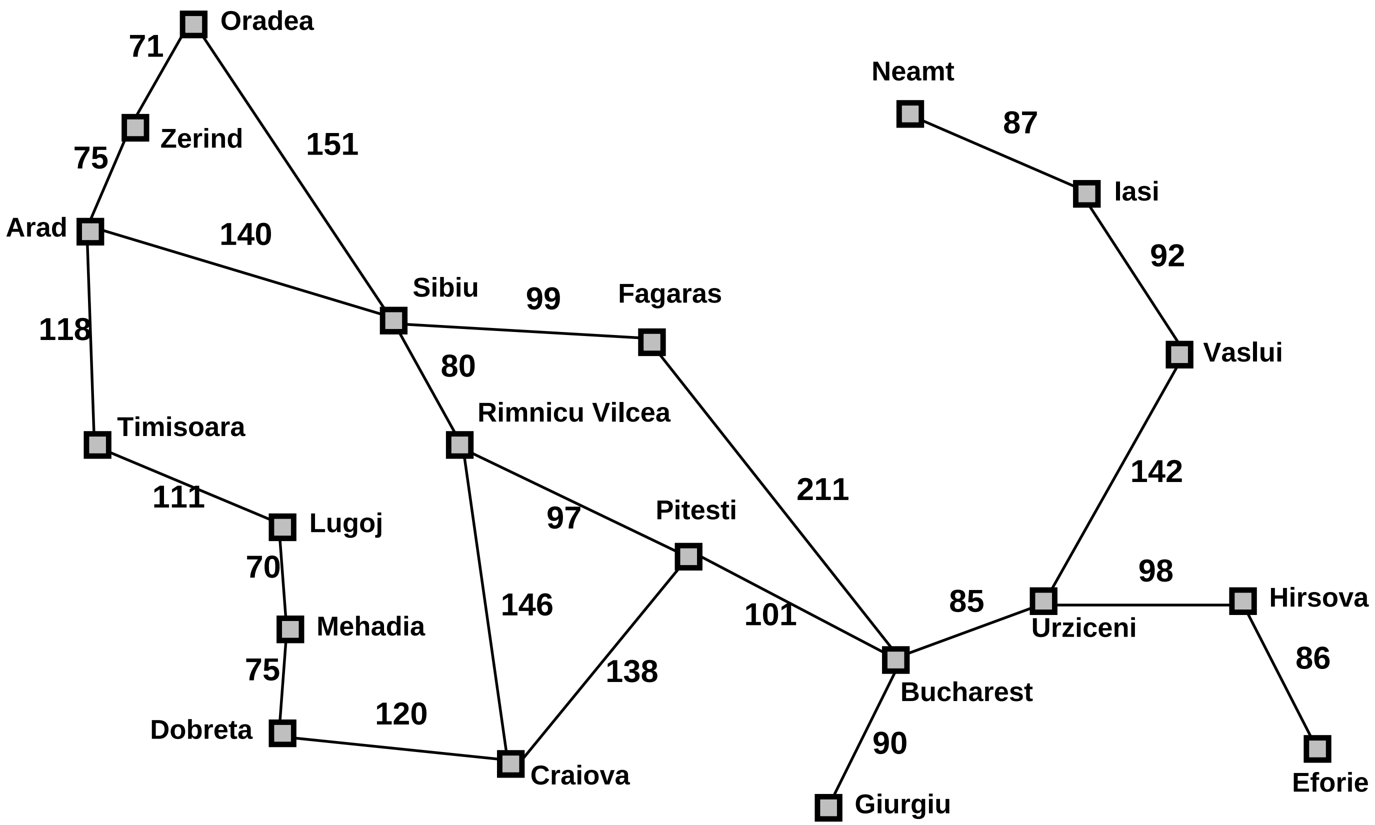
Informed search algorithms

Chapter 3, Sections 5–6

Straight Line Distance



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

Outline

- ♢ Evaluation functions
- ♢ Greedy search
- ♢ A^* search
- ♢ Designing heuristics
- ♢ Graph search

Review: Tree search

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

// Recall Your Haskell Assignment

Note: the goal test is performed when the node is popped from the frontier, **NOT when it is generated during expansion.** This is important when looking for optimal solutions. // So A* does not use them

A strategy is defined by picking the **order of node expansion**

Evaluation function

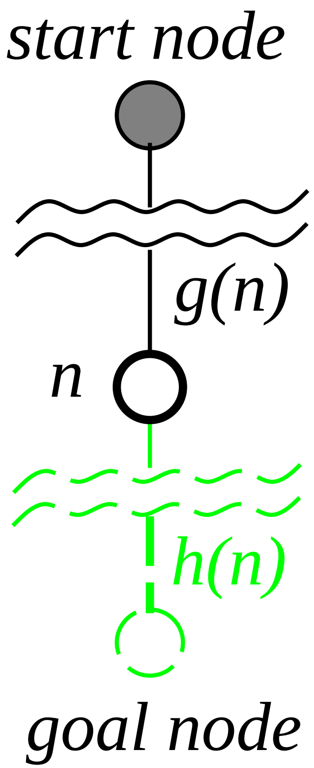
Evaluation function $f(n) = g(n) + h(n)$
– estimate of "desirability", usually problem-specific.

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

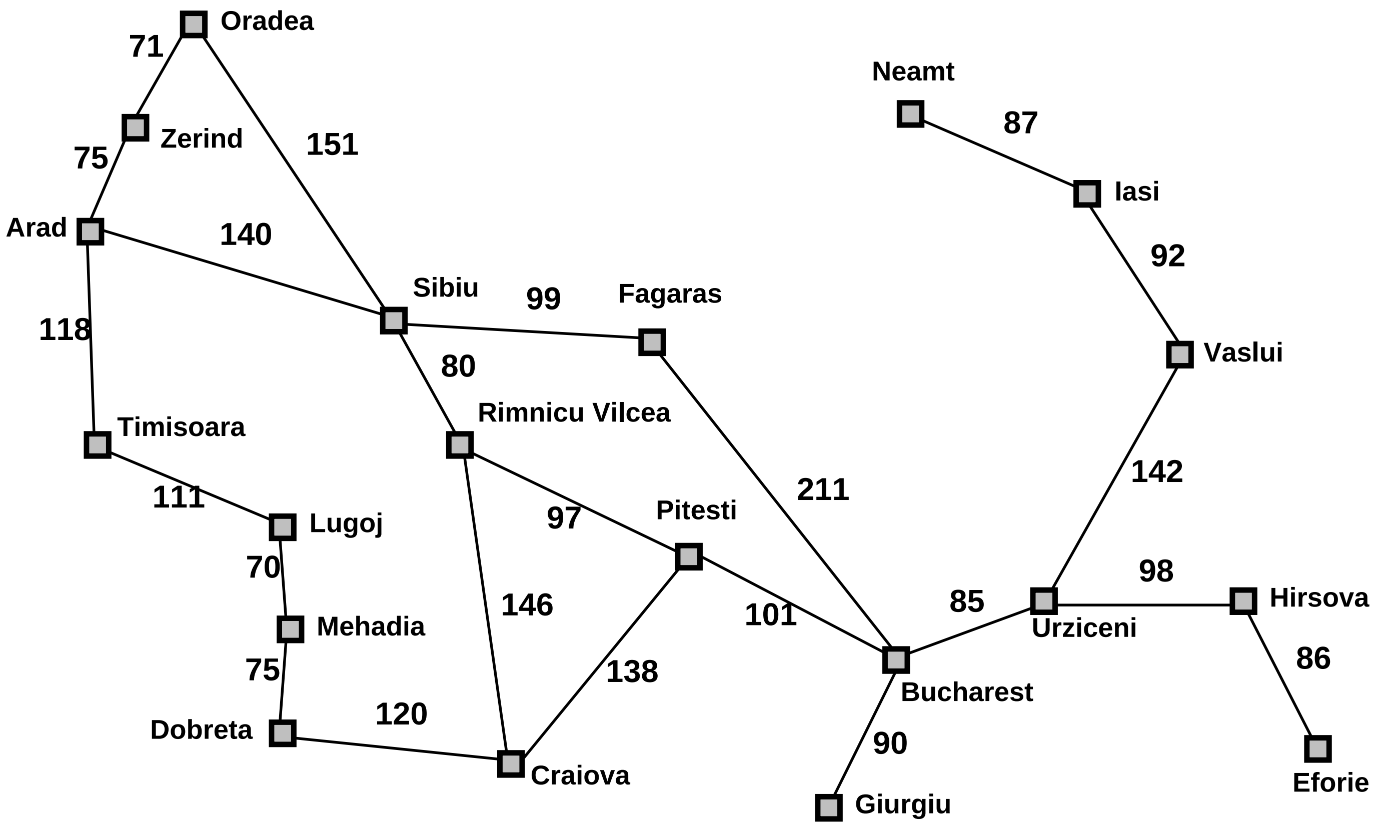
The lower $f(n)$, the more desirable n is // cost mix

Implementation:
frontier is a queue sorted in ascending value of $f(n)$

Special cases:
uniform cost search (uninformed): $f(n) = g(n)$
greedy search (informed) $f(n) = h(n)$
A* search (informed) $f(n) = g(n) + h(n)$



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

Greedy search

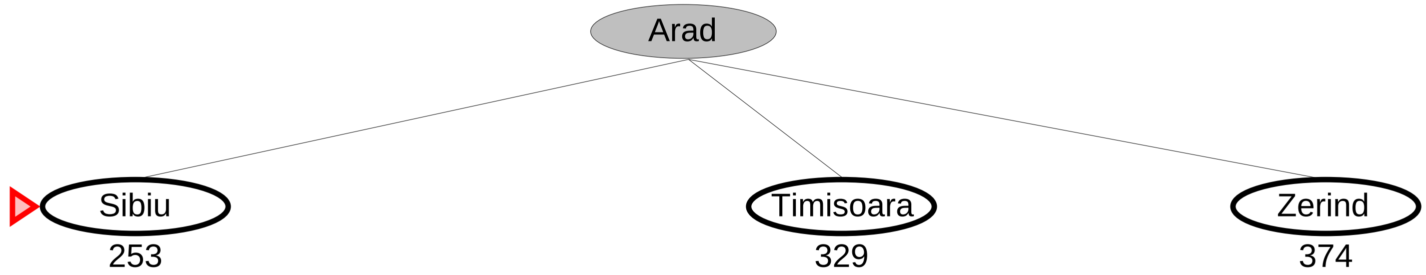
Evaluation function $f(n) = h(n)$ (entirely heuristic)
$\quad = $ estimate of cost from n to the closest goal

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

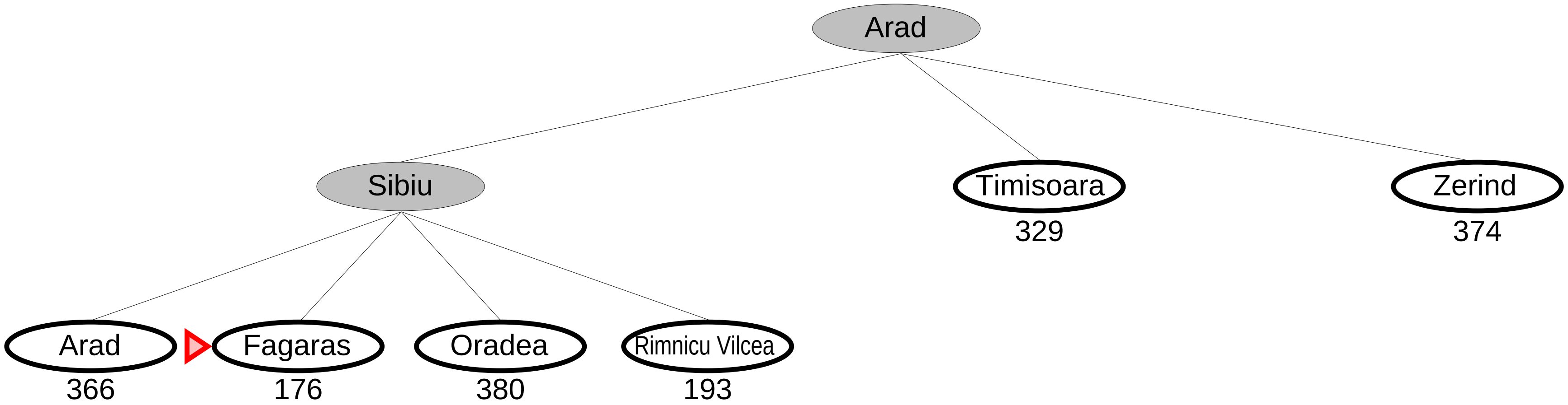
Greedy search example

Greedy search example

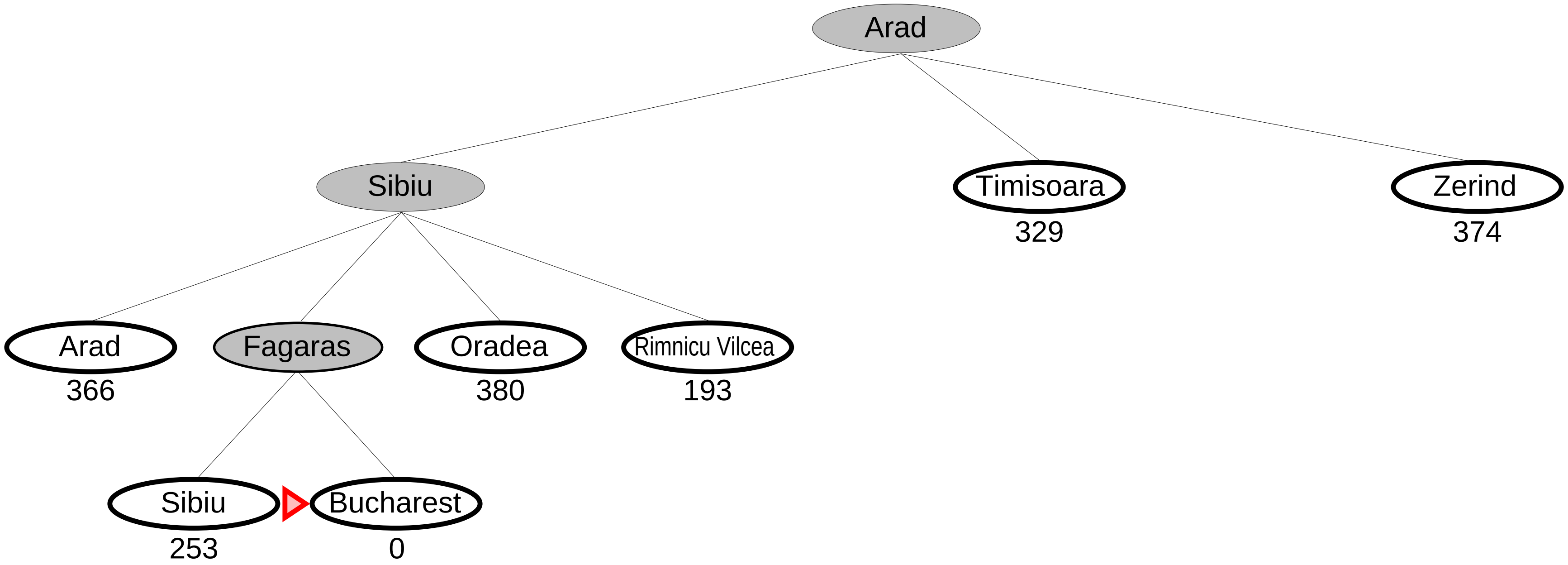


"Pick up the cheapest for each stage"

Greedy search example



Greedy search example



Properties of greedy search

Complete??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with going from Iasi to Fagaras,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with going from Iasi to Fagaras,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

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

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with going from Iasi to Fagaras,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

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

Space?? $O(b^m)$

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with going from Iasi to Fagaras,

Iasi $\to$ Neamt $\to$ Iasi $\to$ Neamt $\to$

Complete in finite space with repeated-state checking

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

enabling sth./sb. to discover/learn sth. for themselves

In CS. it refers to "shortcut" methods (compared to the classic ones). They trade optimality, completeness, accuracy or precision for speed or availability [whether an exact solution can be provided]

Space?? $O(b^m)$

Optimal?? No

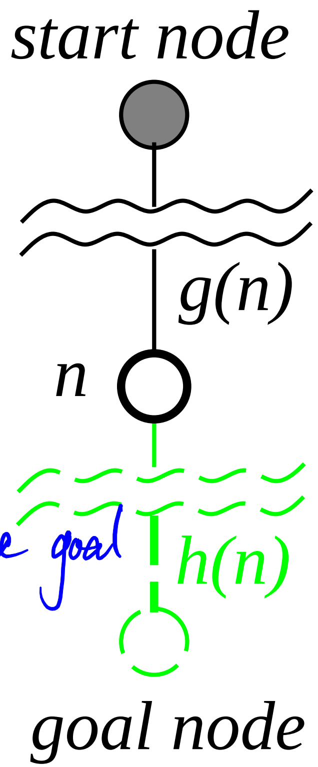
quick, but cannot guarantee optimal result [sometimes even cause loops]

here, it refers to methods that rank search alternatives at each step so as to decide which steps to follow.

estimate of "desirability", usually problem-specific

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 the closest goal
$f(n)$ = estimated total cost of path through n to goal

Admissible heuristic: ≪ Never overestimate the true cost to the goal
$\forall n \; h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

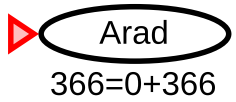
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

When $h(n)$ is admissible, $f(n)$ never overestimates the total cost of the shortest path through n to the goal

Theorem: if h is admissible, A* search finds the optimal solution

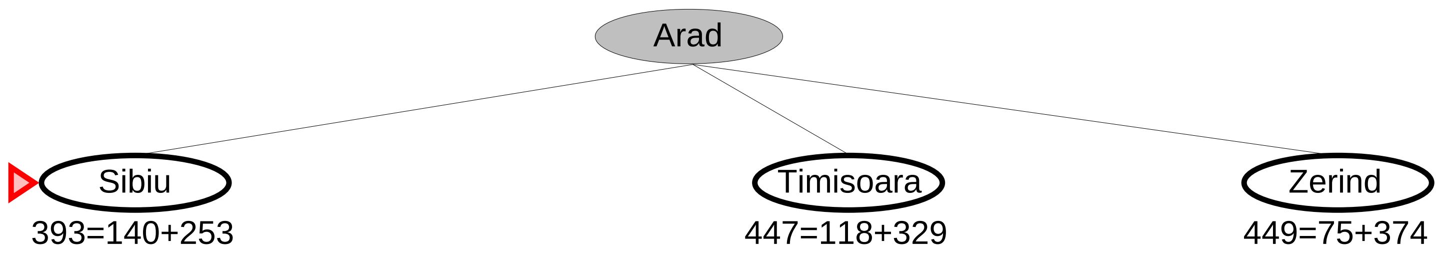
frontier ⇒ a queue sorted in ascending value of $f(n)$

A* search example

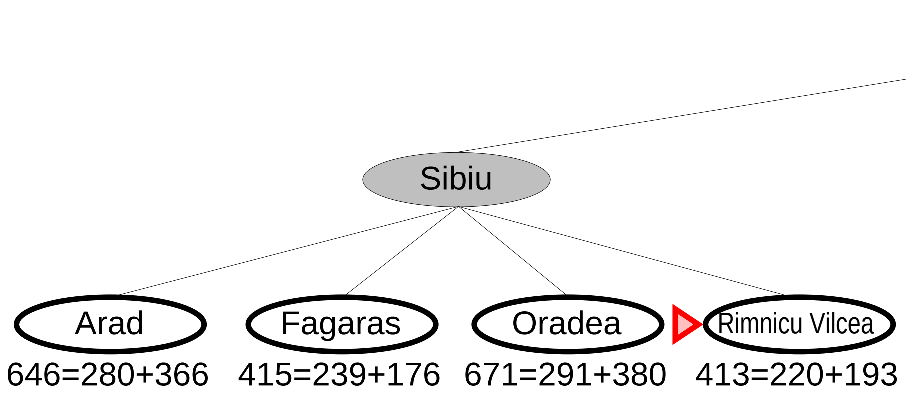


[g + h]

A* search example

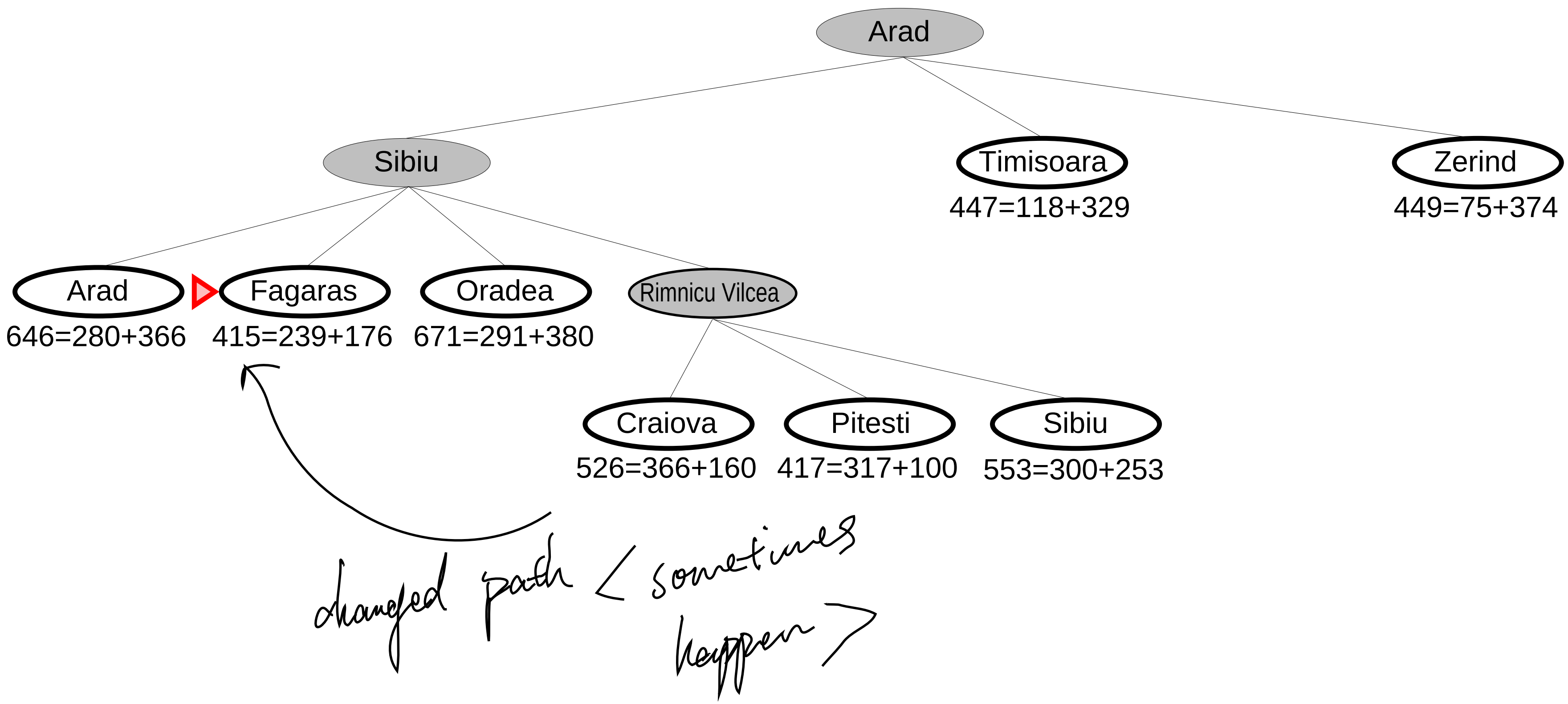


A* search example

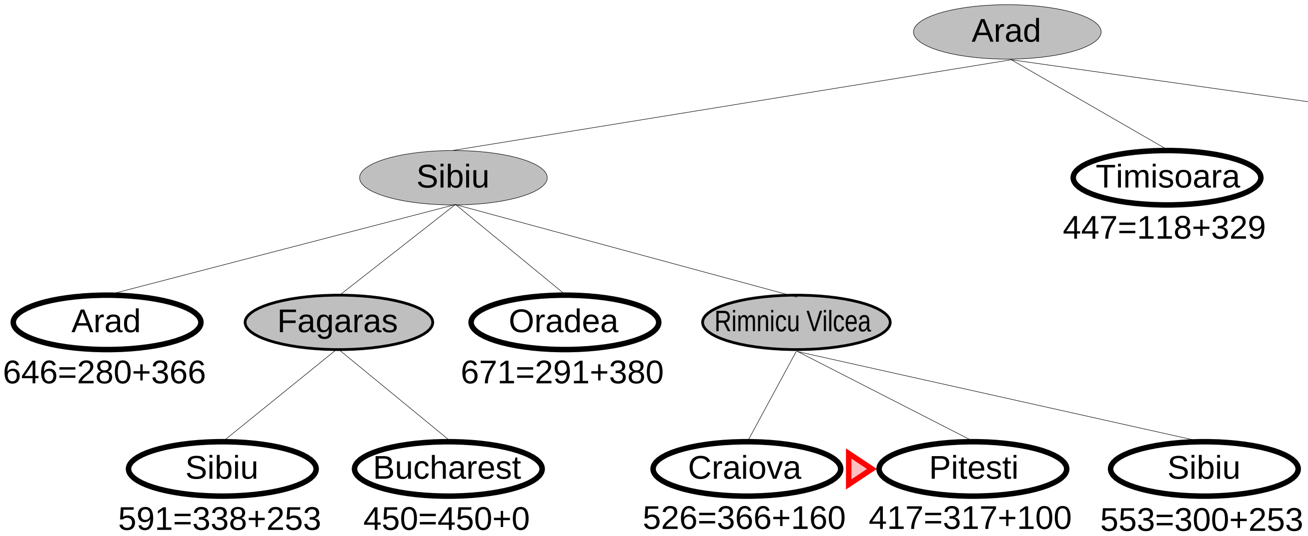


/ taken account into where we came from

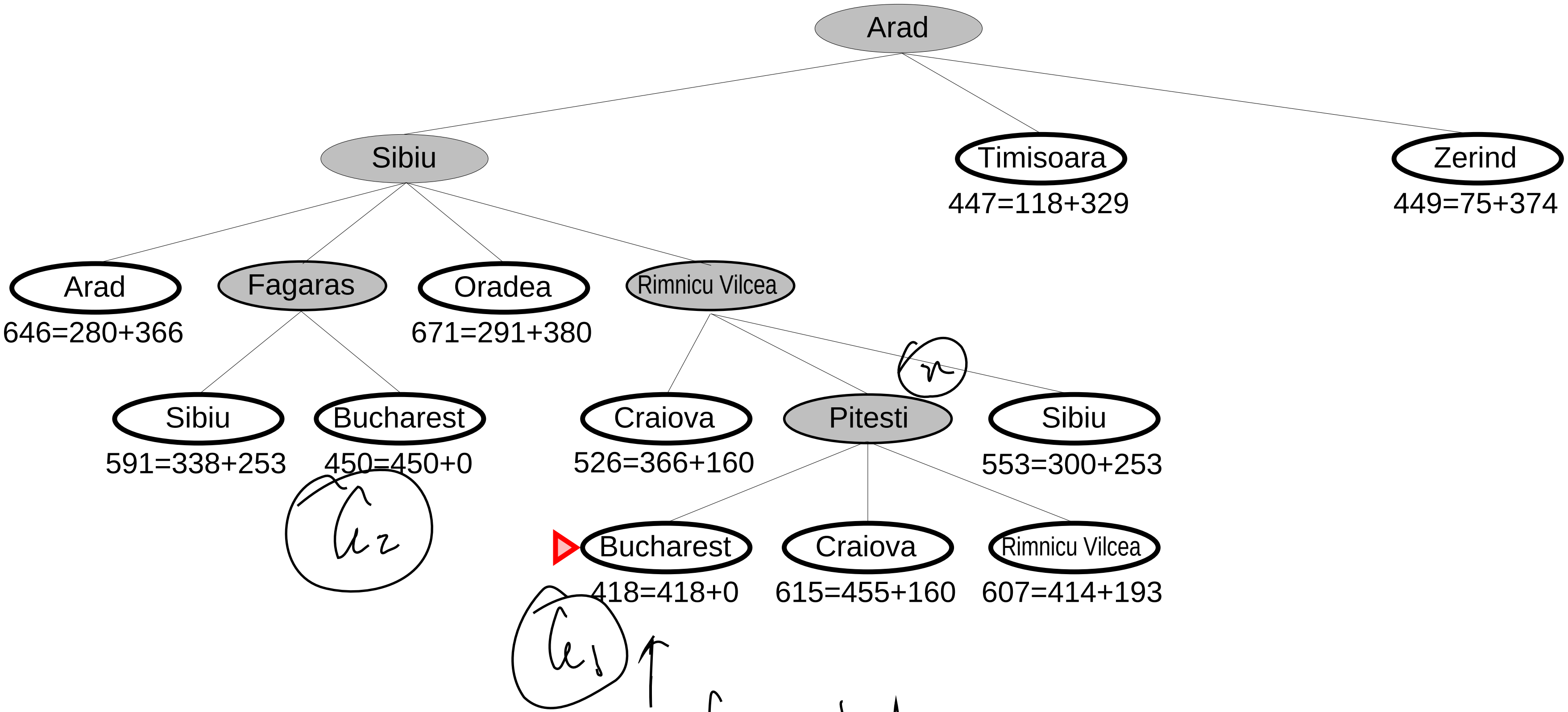
A* search example



A* search example



A* search example

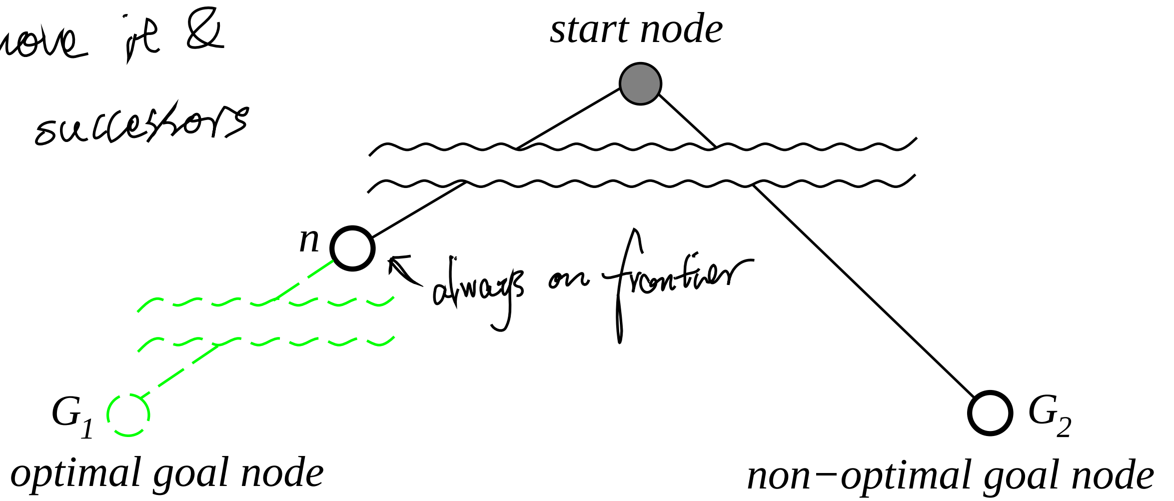


optimal reached

Optimality of A* (based on admissibility)

Suppose some suboptimal goal G_2 has been generated and is in the frontier.
Let n be a frontier node on a shortest path to an optimal goal G_1.

everytime we remove it & replace with its successors



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

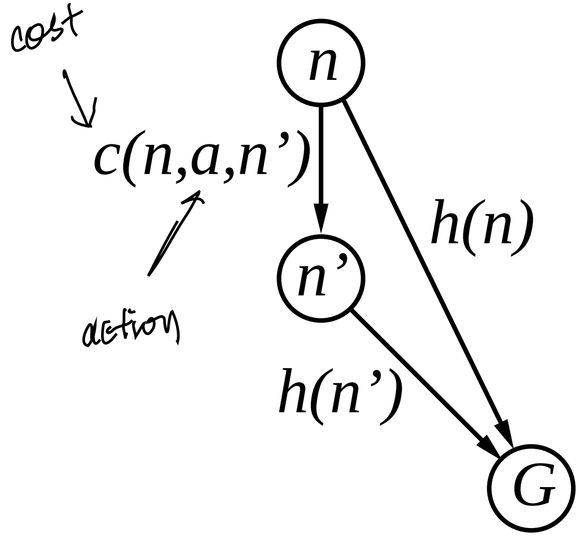
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible, } f(n) \text{ does not overestimate } g(G_1)
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Consistency

A heuristic is consistent if

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



If h is consistent, then h is admissible, and $f(n)$ is nondecreasing along any path:

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

Consequently, when expanding a node, we cannot get a node with a smaller f, and so the value of the best node on the frontier will never decrease.

Once a node is expanded, the cost by which it was reached is the lowest possible.

$f(n)$

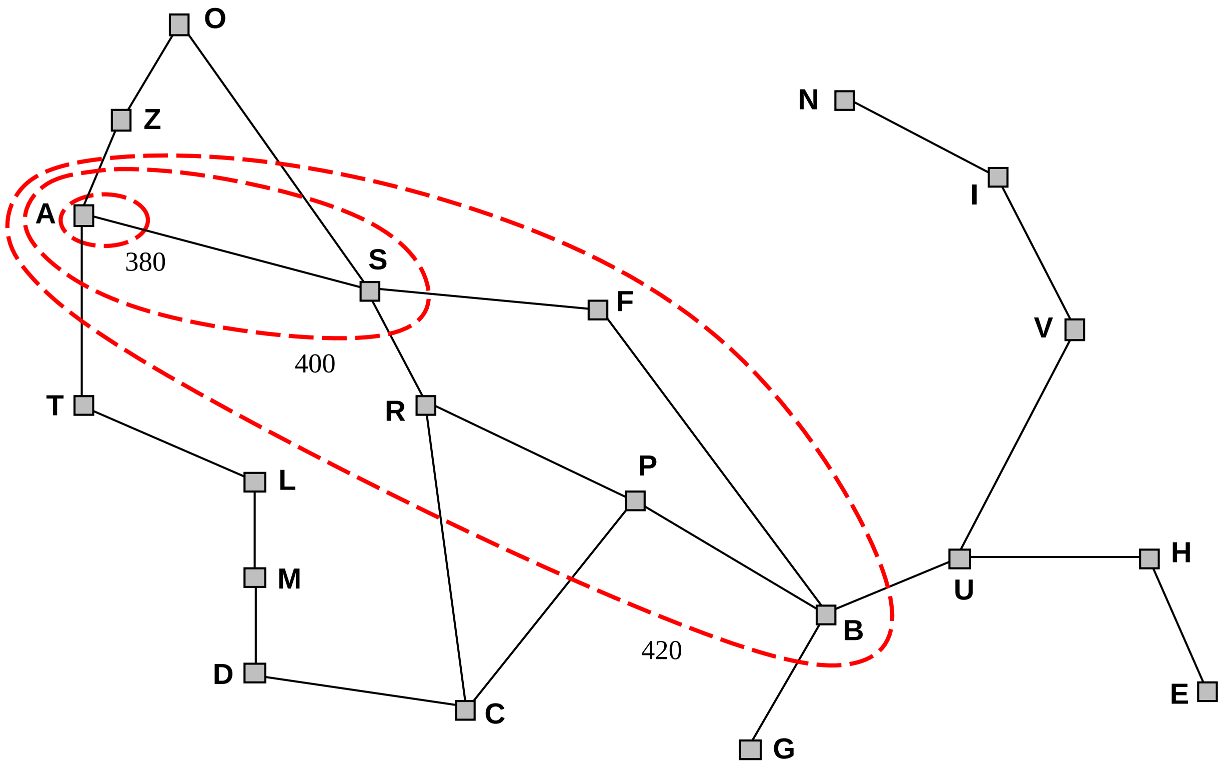
Optimality of A* (based on consistency)

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

Gradually expands "f-contours" of nodes

Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$

(breadth-first expands layers, uniform-cost expands g-contours)



Properties of A*

Complete??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq C^*$

Time??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq C^*$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq C^*$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Exponential

Optimal??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq C^*$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Exponential

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$ (but only one goal node)
A* expands no nodes with $f(n) > C^*$

IDA* is an iterative deepening version of A*with a cutoff on f

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles < the natural work we have to do

$h_2(n)$ = total Manhattan distance

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

thus not admissible heuristics

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$\underline{h_1(S) =}$??

$\underline{h_2(S) =}$??

$0 \le$ Admissible heuristics $\le$ optimal true cost

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)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

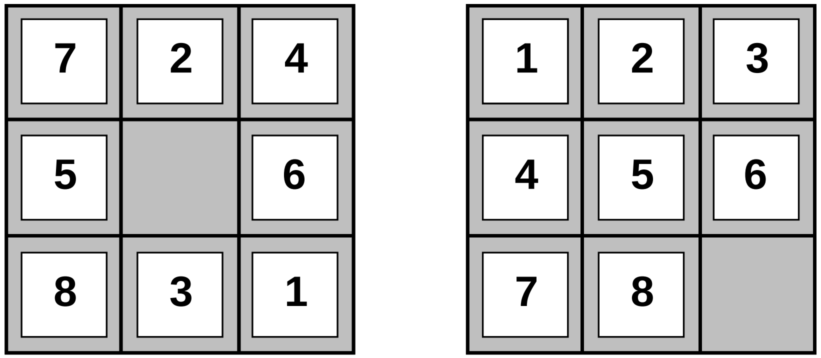
$h_1(S) =$?? 6
$h_2(S) =$?? $\leftarrow$ higher number

thus the higher the less we have to expand

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**

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

Dominance

Given two admissible heuristics h_a, h_b,
if $h_b(n) \geq h_a(n)$ for all n then h_b **dominates** h_a and is better for search

In the 8-puzzle h_2 dominates h_1. Typical search costs:

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

There is a tradeoff between the accuracy of h and the time to compute h

want fast & accurate enough

Given two admissible heuristics h_a, h_b,

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

if neither dominates

combine them

is also admissible and dominates h_a, h_b

Is $h_a(n) + h_b(n)$ admissible??

by taking the max

no higher & dominate both

could double-count things, sometimes okey

Relaxed problems

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

// remove some constrains

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

Rules of the 8-puzzle:

> a tile can move from square A to square B if ① A is adjacent to B and ② B is blank; get all tiles in their correct positions.

If we relax the rules so that a tile can move **anywhere**, then $h_1(n)$ gives the shortest solution

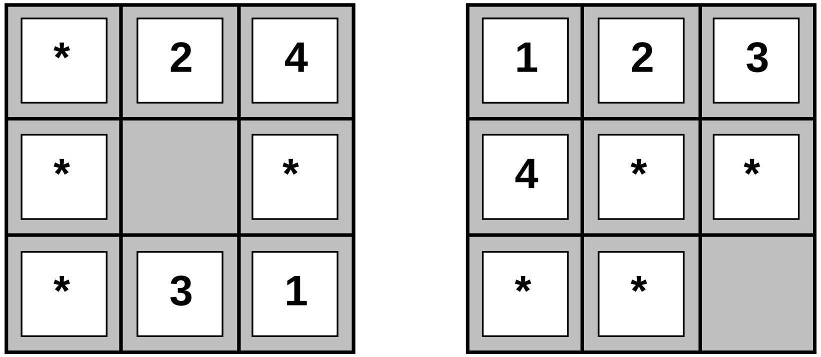
If we relax the rules so that a tile can move to **any adjacent square**, then $h_2(n)$ gives the shortest solution

Relaxed problems

Rules of the 8-puzzle:

> a tile can move from square A to square B if A is adjacent to B and B is blank; get all tiles in their correct positions.

Relaxing the rules so that only **some** tiles need to get in their correct positions and solving the relaxed problem optimally yields another admissible heuristic

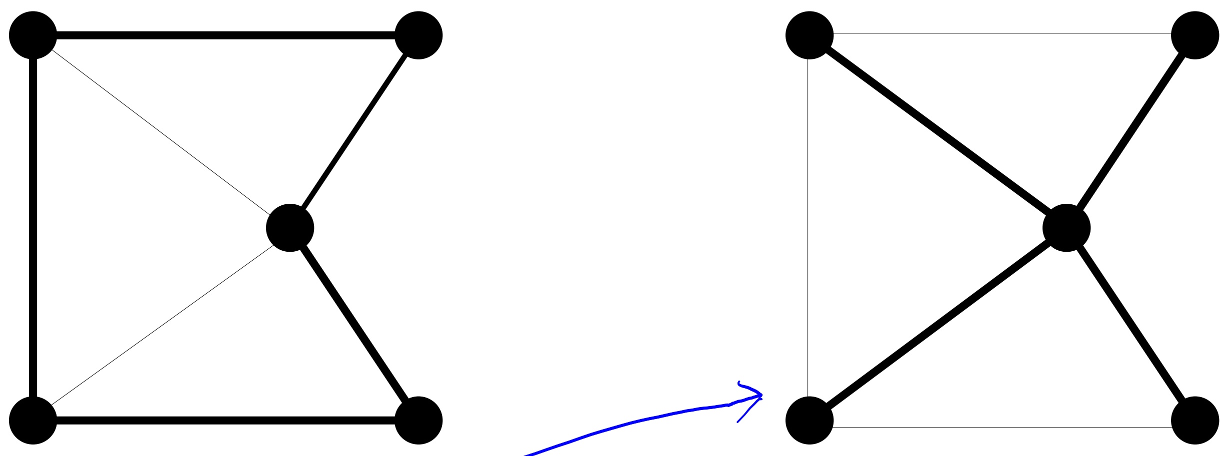


Start State Goal State

Heuristics derived from the cost of an optimal solution to a smaller subproblem are used in pattern databases to store solutions for every possible subproblem up to a given size.

Relaxed problems

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



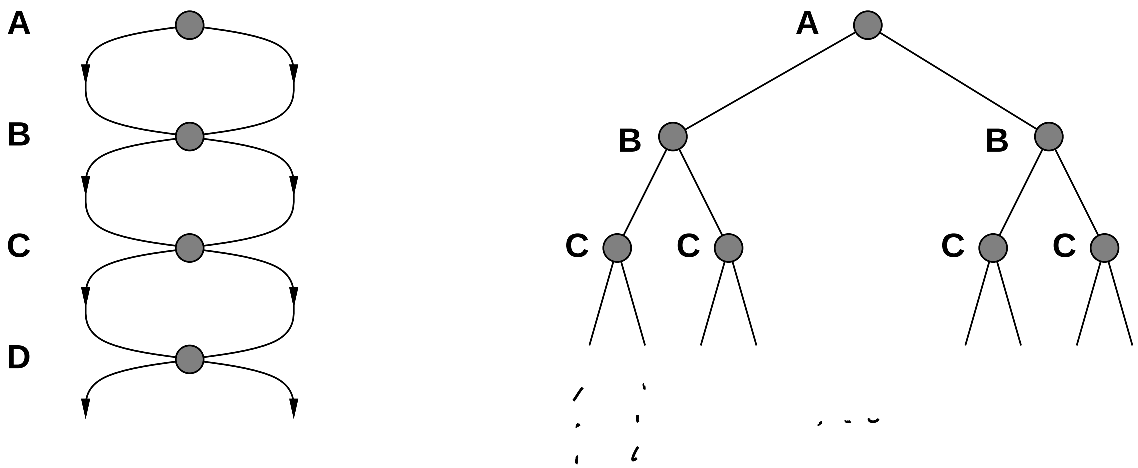
Minimum spanning tree can be computed in $O(n^2)$
and the sum of the edge costs in an MST is a lower bound on the optimal (open) tour cost

Tree Search and Repeated States

◇ For many problems, the state space is a graph rather than a tree

◇ Cycles can prevent termination

◇ Failure to detect repeated states can turn a linear problem into an exponential one!



Graph search

function GRAPH-SEARCH(*problem, frontier*) **returns** a solution, or failure

    explored ← an empty set of nodes
    frontier ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *frontier*)
    loop do
        if *frontier* is empty **then return** failure
        node ← REMOVE-FRONT(*frontier*)
        if GOAL-TEST(*problem*, STATE[*node*]) **then return** *node*
        add *node* to *explored*
        frontier ← INSERTNODES(EXPAND(*node*, *problem*), *frontier*)

---

function INSERTNODES(*nodes, frontier*) **returns** updated frontier

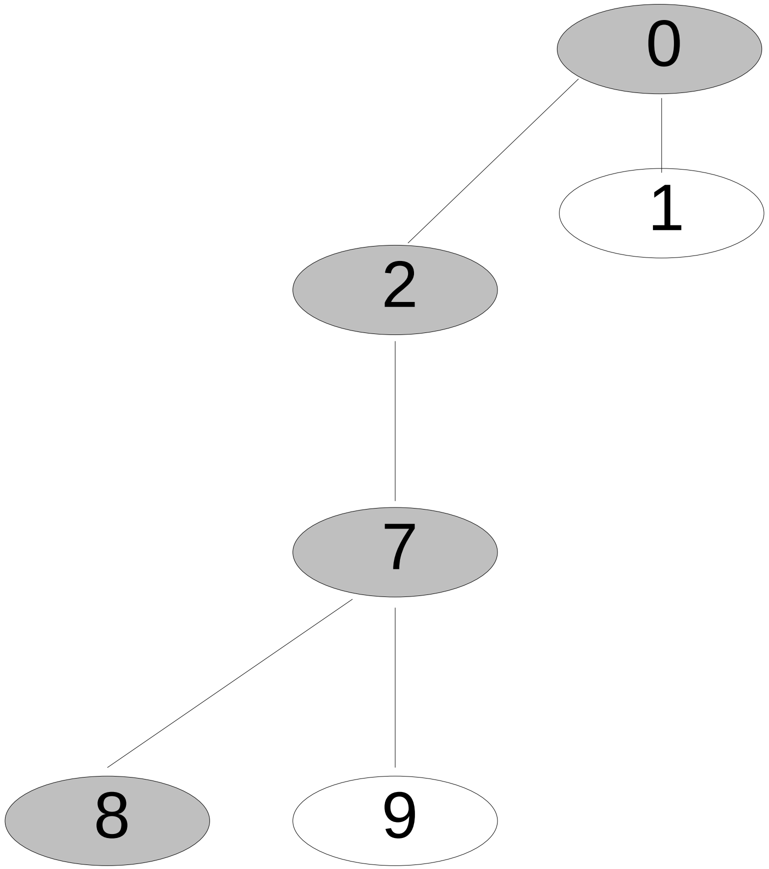
    for each *n* in *nodes* **do**
      if $\nexists$ *m* in *explored* ∪ *frontier* s.t. STATE[*m*] = STATE[*n*] **then**
         add *n* to *frontier*
    return *frontier*

At most one instance of each state in $explored \cup frontier$. All expanded nodes kept in memory!!

exponential memory

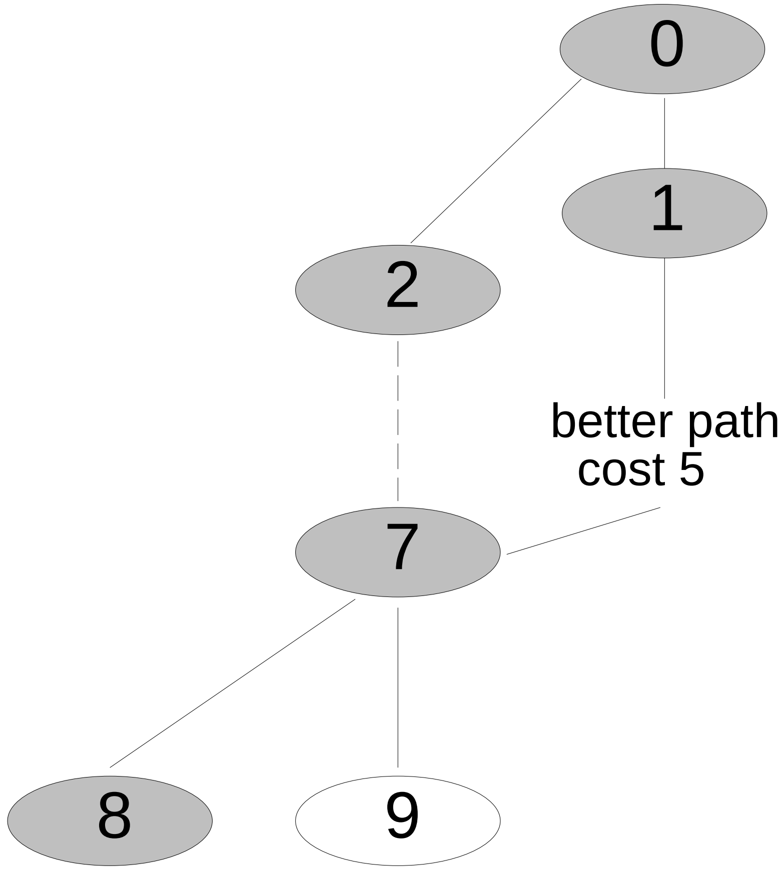
Graph search and Optimality

◊ When seeking optimal solutions, mutliple paths to the same state may need to be explored and compared to find the optimal



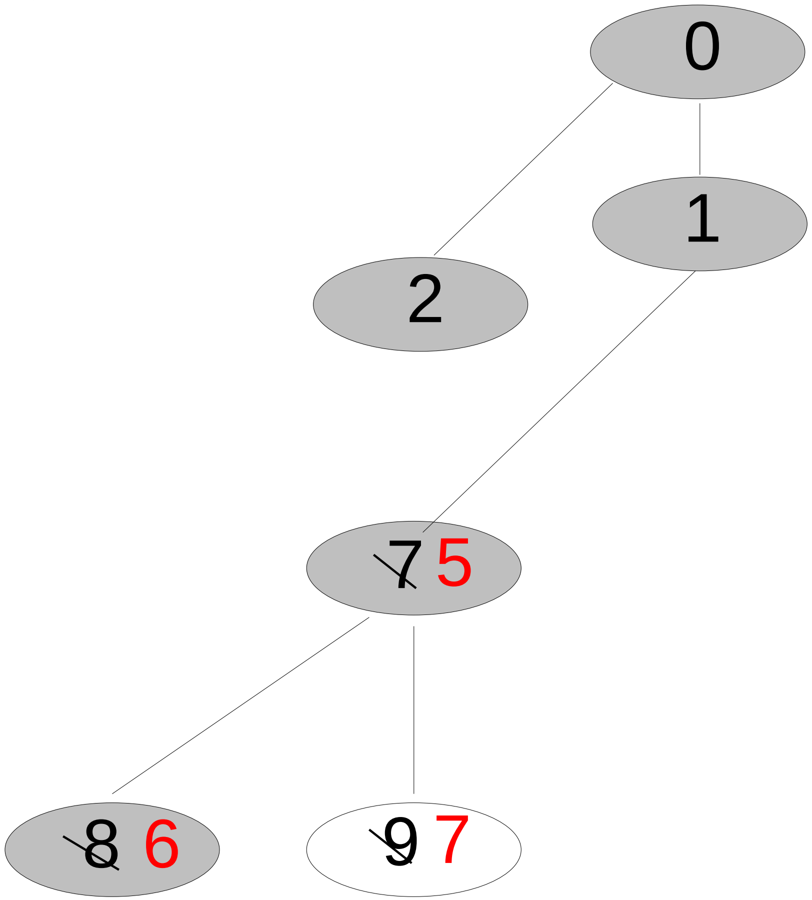
Graph search and Optimality

◇ When seeking optimal solutions, mutliple paths to the same state may need to be explored and compared to find the optimal



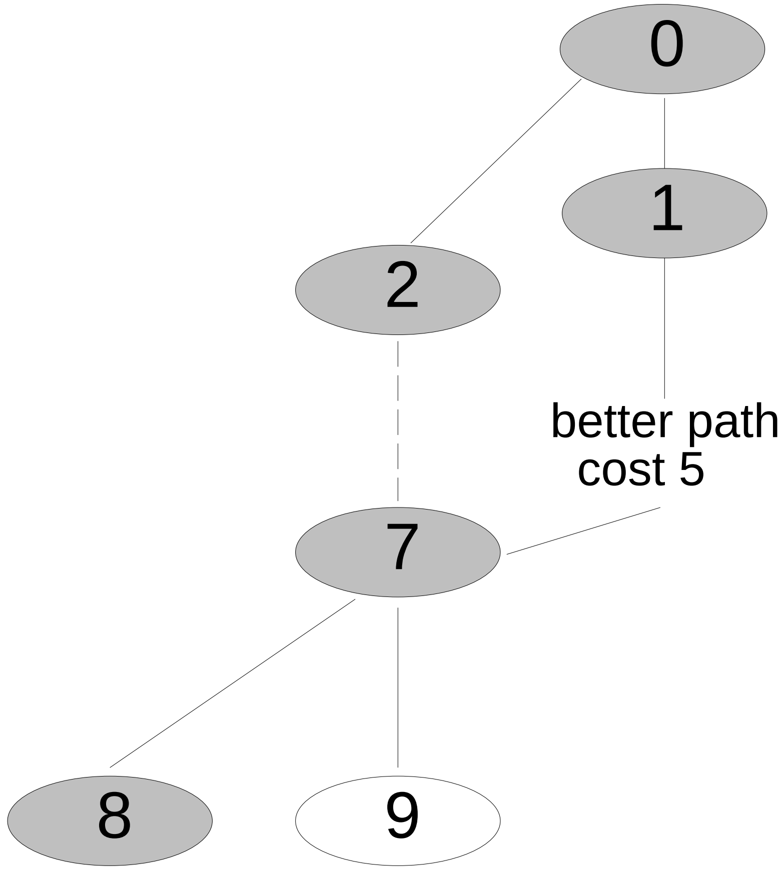
Graph search and Optimality

♢ We may need to keep the better node and update the depths and path-costs of the descendants of the worst one.



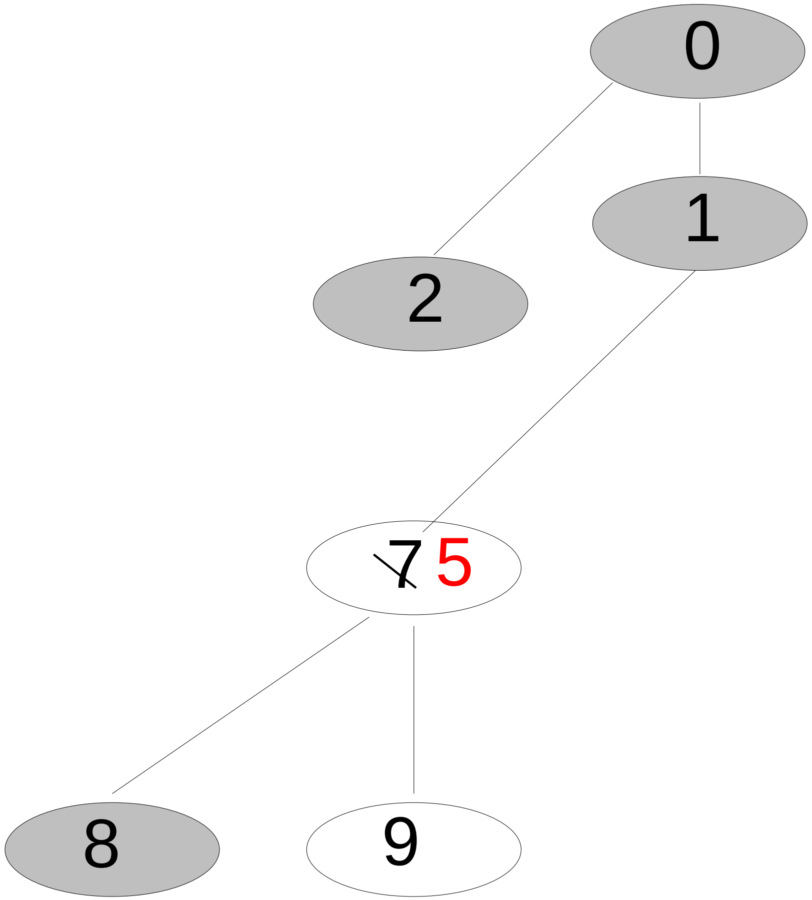
Graph search and Optimality

♢ Trick to avoid updating descendants: re-open the explored node; its descendants will be updated when it is re-expanded.



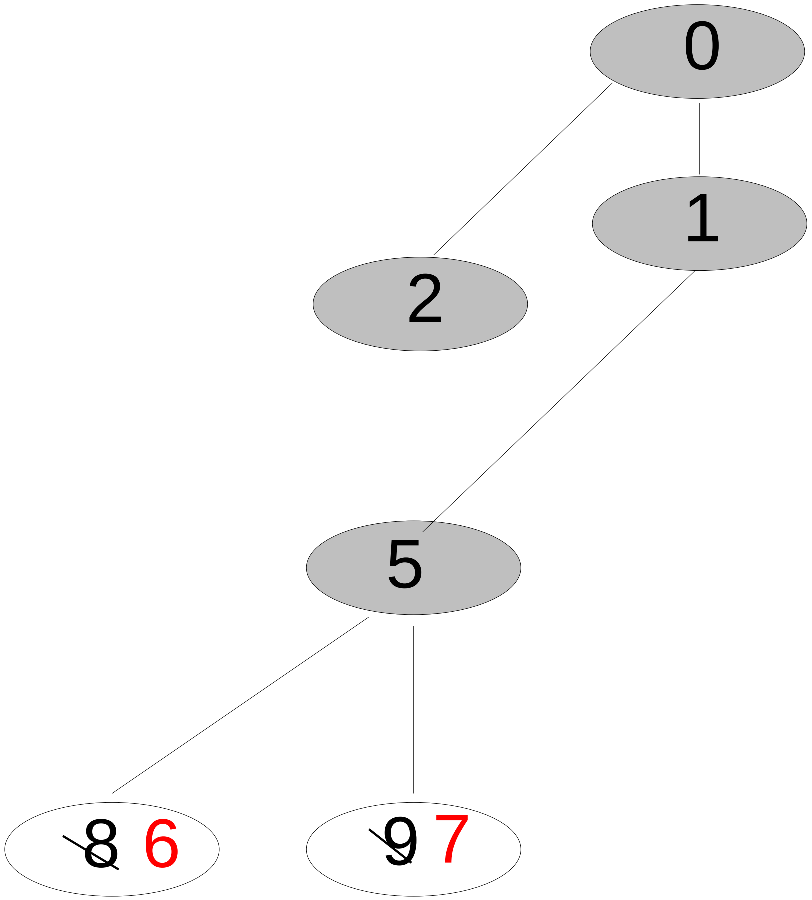
Graph search and Optimality

♢ Trick to avoid updating descendants: re-open the explored node; its descendants will be updated when it is re-expanded.



Graph search and Optimality

◇ Trick to avoid updating descendants: re-open the explored node; its descendants will be updated when it is re-expanded.



※ it will re-open a previously expanded node.

Graph search and Optimality

```
function InsertNodes( nodes, frontier) returns updated frontier
   for each n in nodes do
     if ∄ m in explored ∪ frontier s.t. State[m] = State[n] then
       add n to frontier
     else if Path-Cost[n] < Path-Cost[m]
         Path-Cost[m] ← Path-Cost[n]
         Parent[m] ← Parent[n]
         Action[m] ← Action[n]
         Depth[m] ← Depth[n]
         if m in explored then
           move m back to frontier    // reopen m
   return frontier
```

The extra square is needed with A*.

It is also needed with uniform cost unless step costs are equal.

If h is consistent (not just admissible), no re-opening (last 2 lines) is needed.

$h = 0$ is consistent so no reopening is needed with uniform cost.

Which algorithm and strategy to use

strategy	solution	useful when	frontier	algorithm and state space
DFS	arbitrary	many solutions exist	LIFO	tree-search for finite acyclic graphs graph-search (simple) for finite graphs
BFS	shortest	shallow solutions exist	FIFO	tree search graph search (simple) may improve performance
UC	optimal	good admissible heuristic lacking	priority queue ordered by g	tree search for trees graph-search (simple) for equal step costs graph-search (optimal) for arbitrary step costs (no reopening needed)
A*	optimal	good admissible heuristics exist	priority queue ordered by f	tree search for trees and admissible heuristics graph-search (optimal) for admissible heuristics (no reopening needed for consistent heuristics)
greedy search	arbitrary but maybe good	good (inadmissible) heuristics exist	priority queue ordered by h	tree search for finite acyclic graphs graph search (simple) for finite graphs

Summary

- **Heuristic functions** estimate costs of shortest paths
  – good heuristics can dramatically reduce search cost

- **Greedy best-first** search expands lowest h
  – incomplete and not always optimal

- **A*** search expands lowest $g + h$
  – complete and optimal

- **Admissible heuristics** underestimate the optimal cost
  – they can be derived from exact solutions to relaxed problems

- **Graph search** can be exponentially more efficient than tree search
  – often needed to ensure termination and optimality
  – stores all expanded nodes and requires extra tests