Informed Search Algorithms

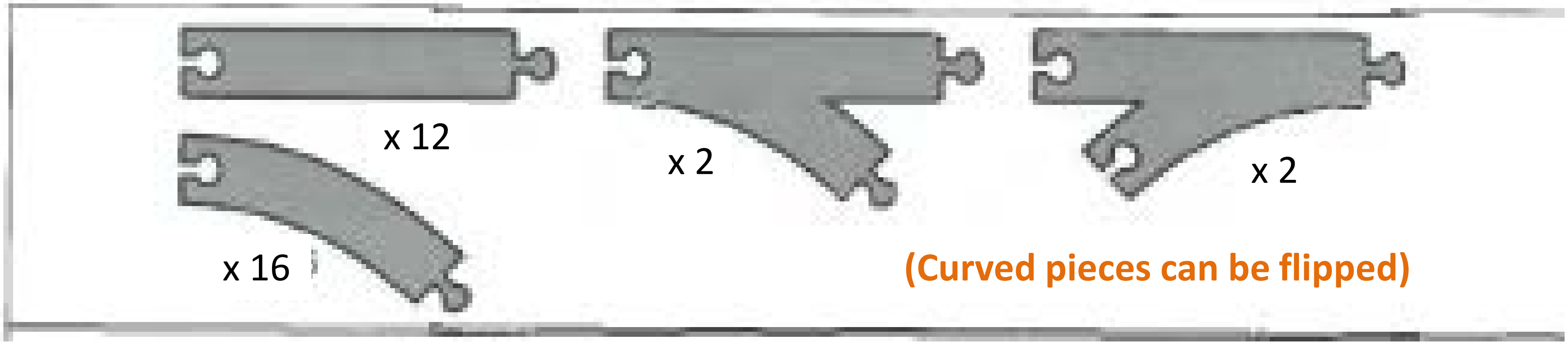
Chapter 3, Sections 3.5-3.6

Reading Assignment

- For Thursday: Chapter 4.3-4.5 (we’re skipping 4.1-4.2)
- For next week: Chapter 5

Class Exercise: Wooden Railway Set

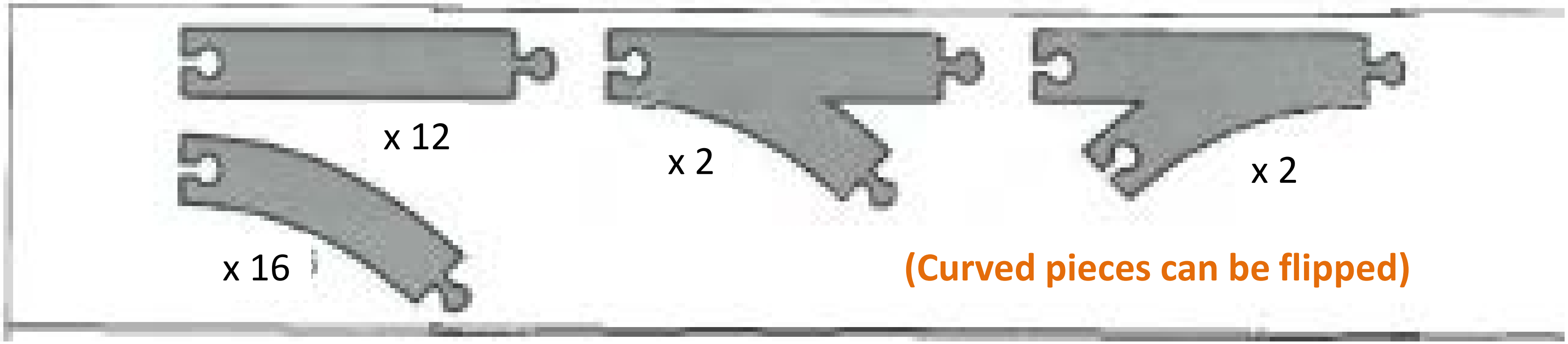
Track pieces from wooden railway set:



Q1: Suppose the pieces fit together exactly. Give formulation of the task as a search problem

Class Exercise: Wooden Railway Set (con’t.)

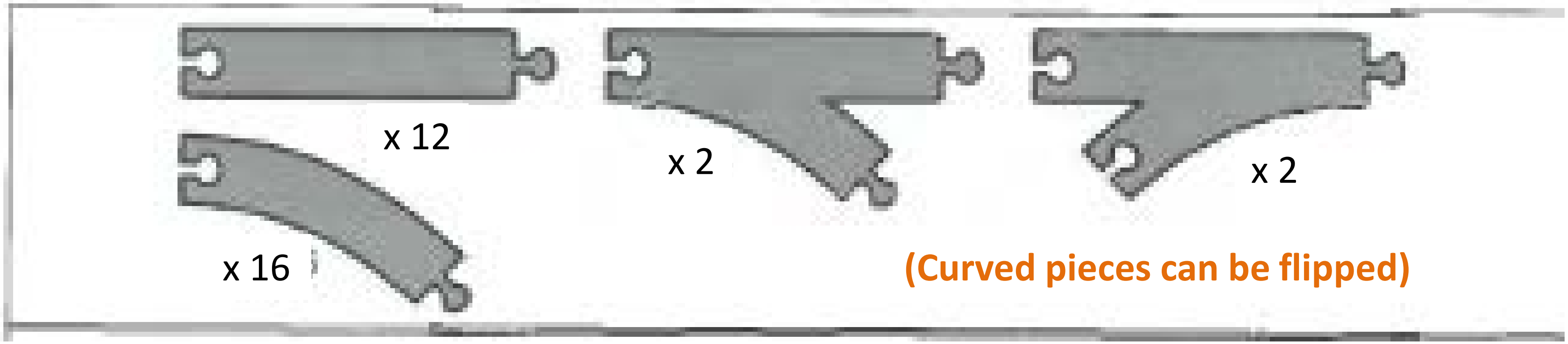
Track pieces from wooden railway set:



Q2: Identify a suitable uninformed search algorithm for this task, and explain why it is suitable.

Class Exercise: Wooden Railway Set (con't.)

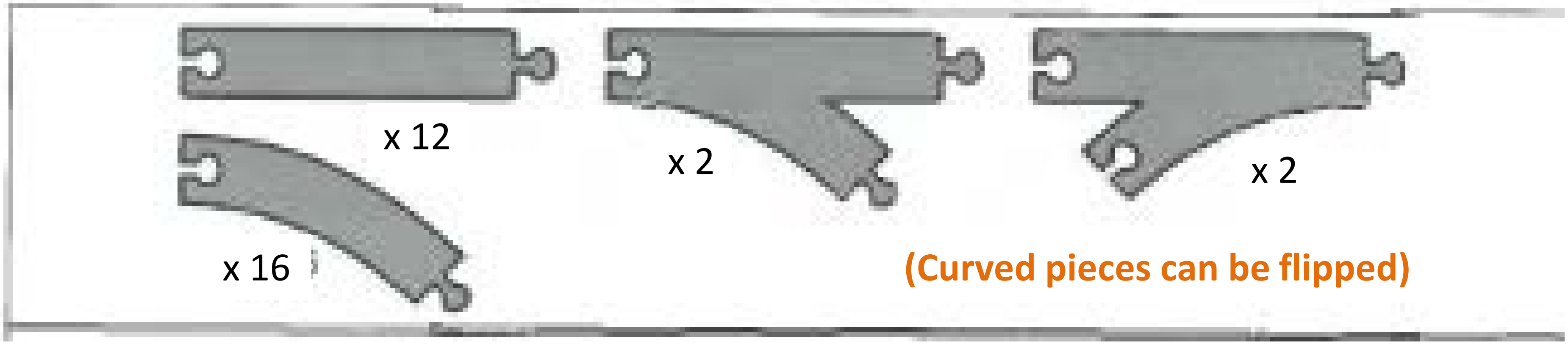
Track pieces from wooden railway set:



Q3: Why does removing any one of the "fork" pieces make the problem unsolvable?

Class Exercise: Wooden Railway Set (con't.)

Track pieces from wooden railway set:



Q4: Give an upper bound on the total size of the state space defined for this formulation. (Ignore problem of overlapping pieces and loose ends. Reason primarily about max branching factor and max depth. Pretending unique pieces.)

Review: Tree search

```
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] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

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

Best-first search

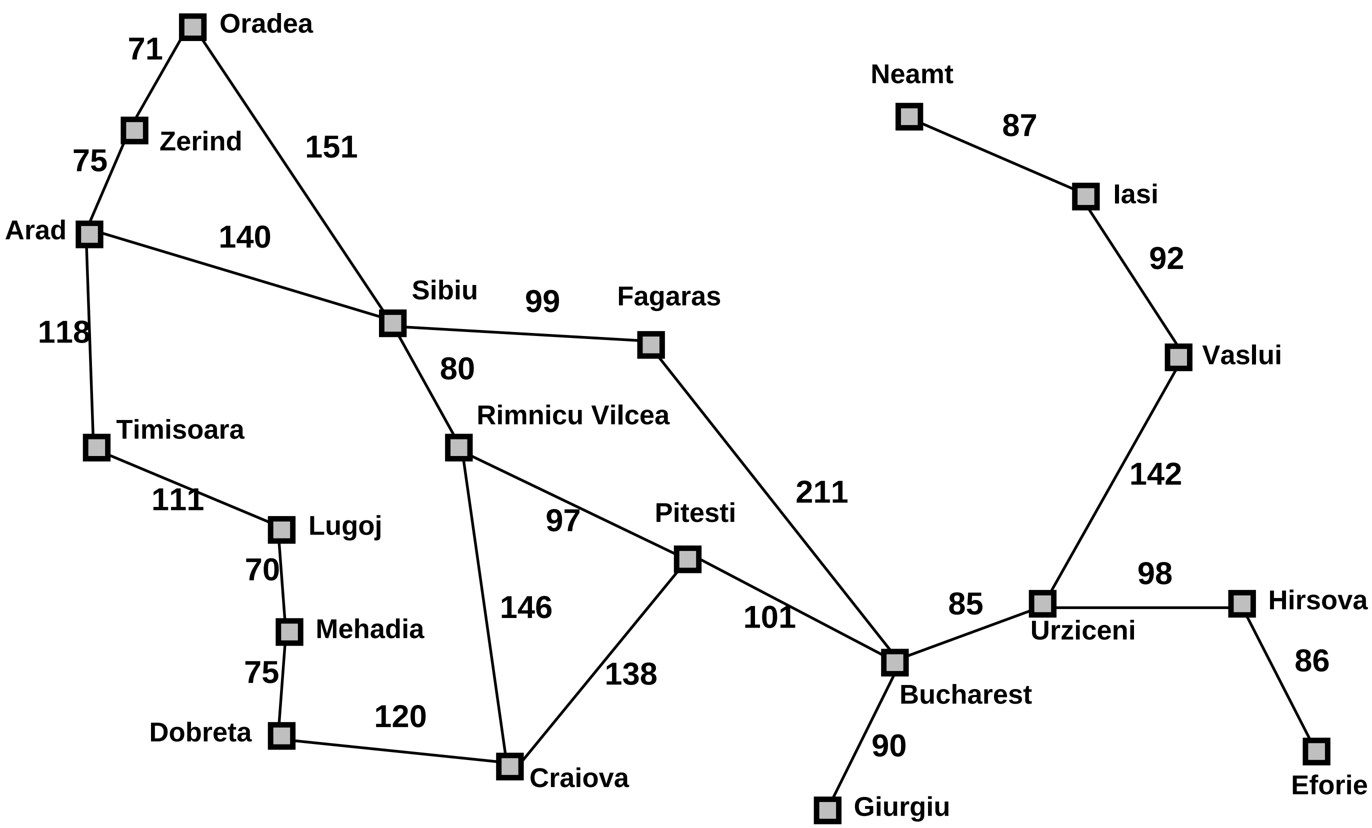
Idea: use an evaluation function for each node
– estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

Implementation:
$fringe$ is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A* search

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

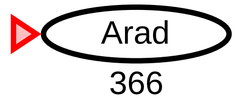
Evaluation function $h(n)$ (**h**euristic)

$= \text{estimate of cost from } n \text{ to the closest goal}$

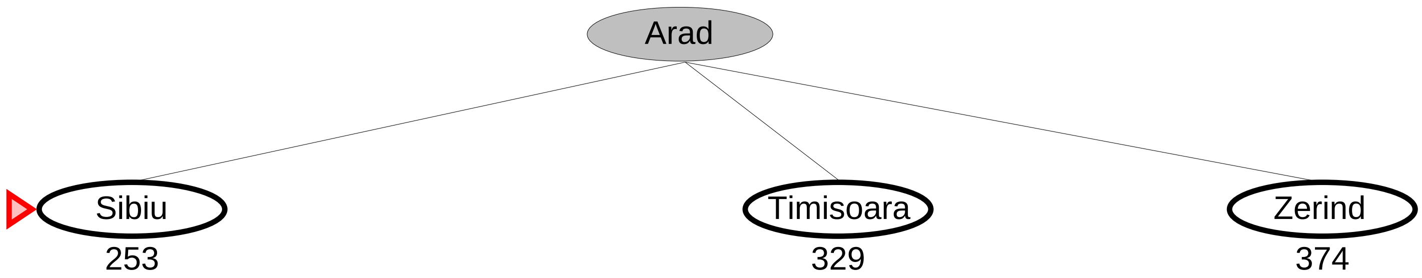
E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

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

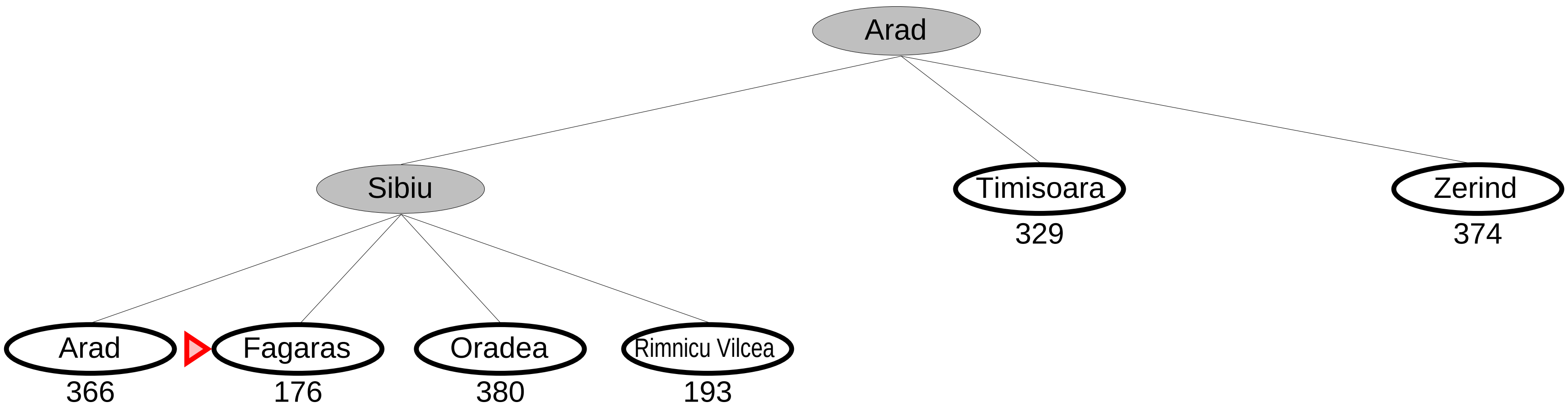
Greedy search example



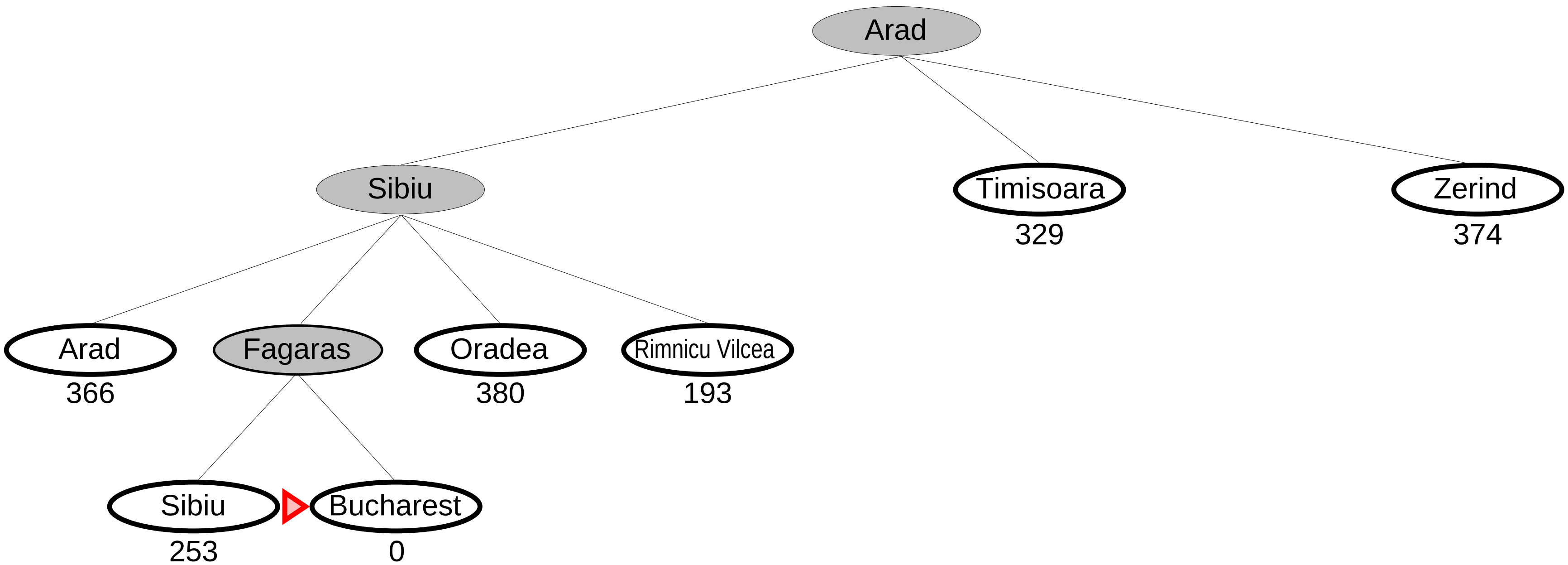
Greedy search example



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 Oradea as goal,

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.,

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.,

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?? $O(b^m)$—keeps all nodes in memory

Optimal??

Properties of greedy search

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

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?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

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

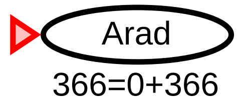
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

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.)

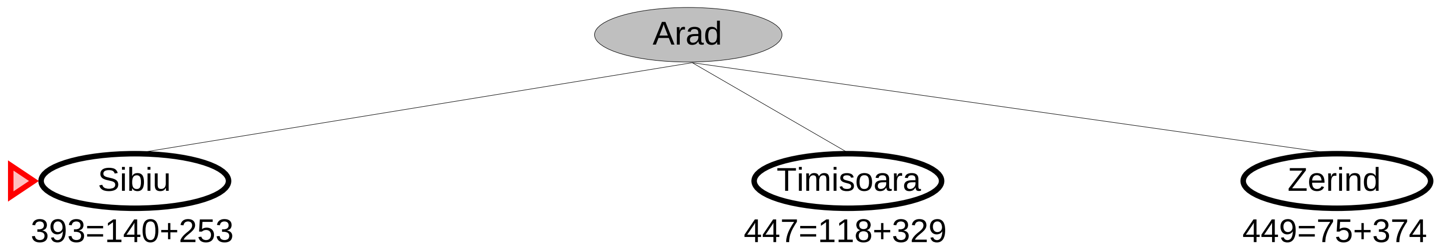
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

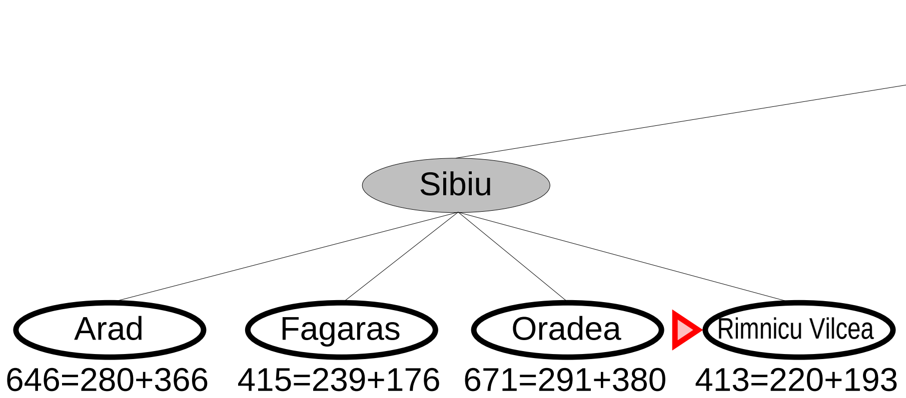
A* search example



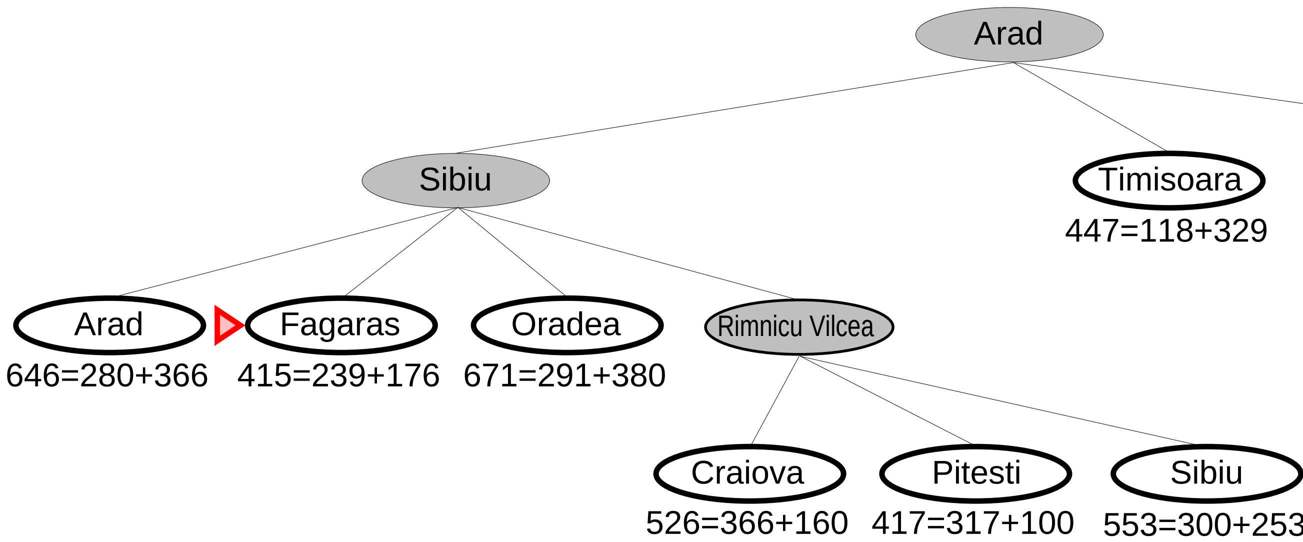
A* search example



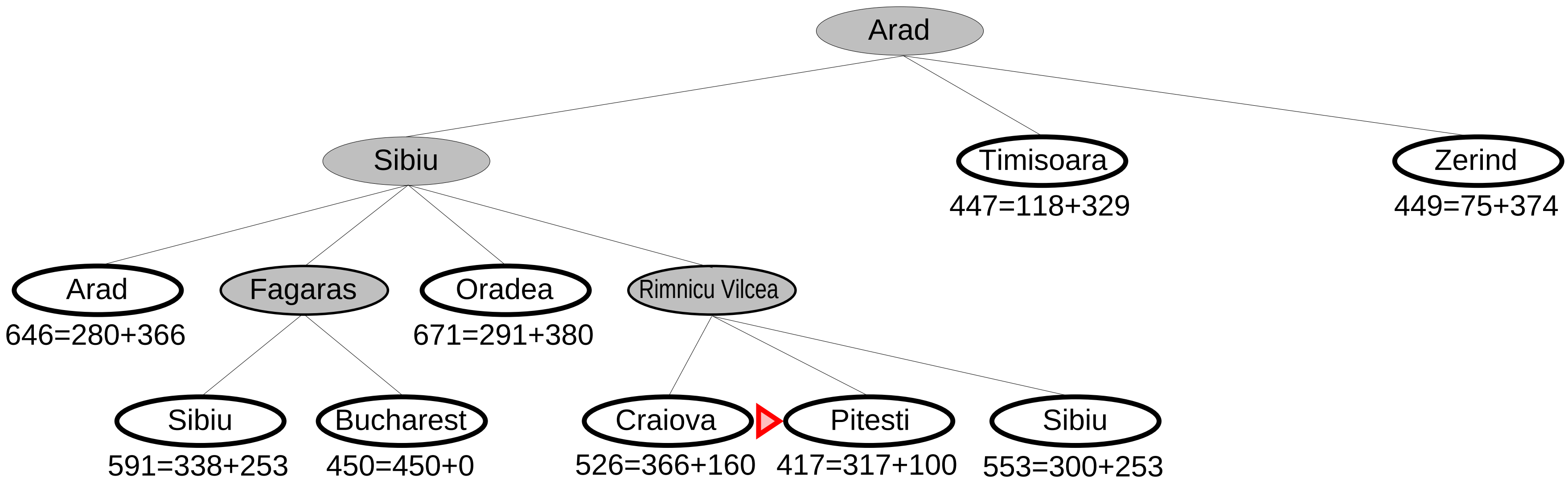
A* search example



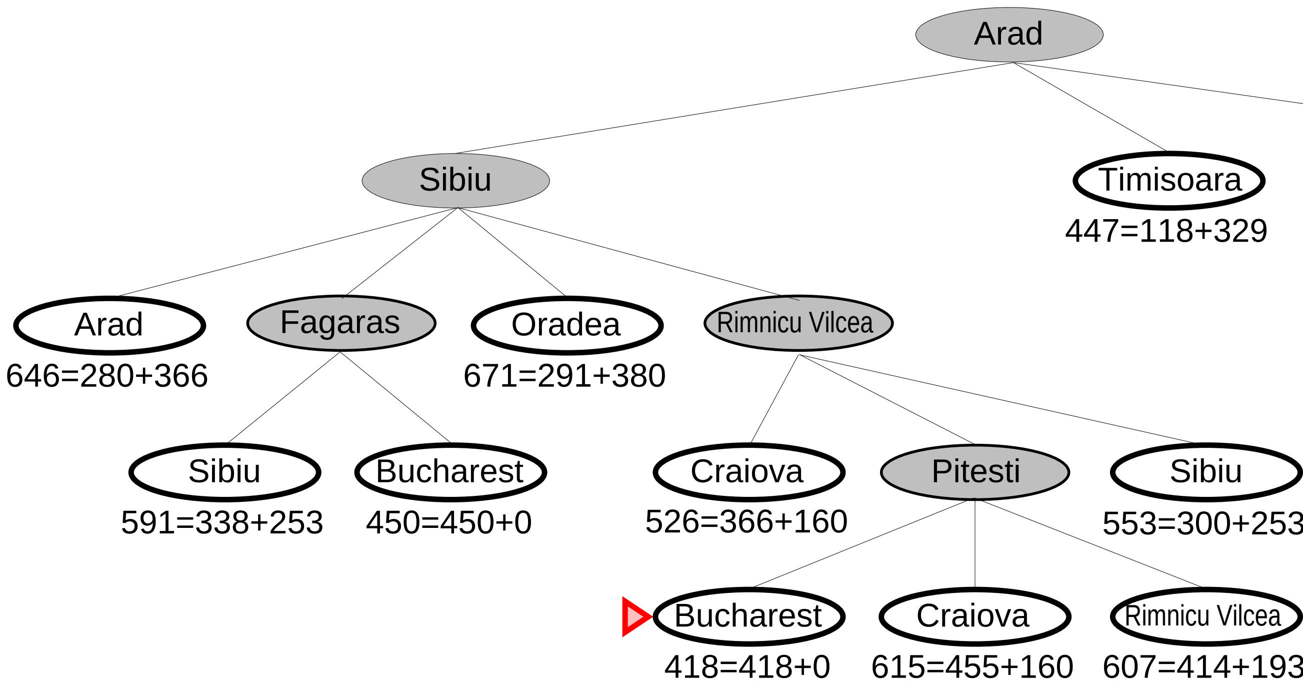
A* search example



A^* search example

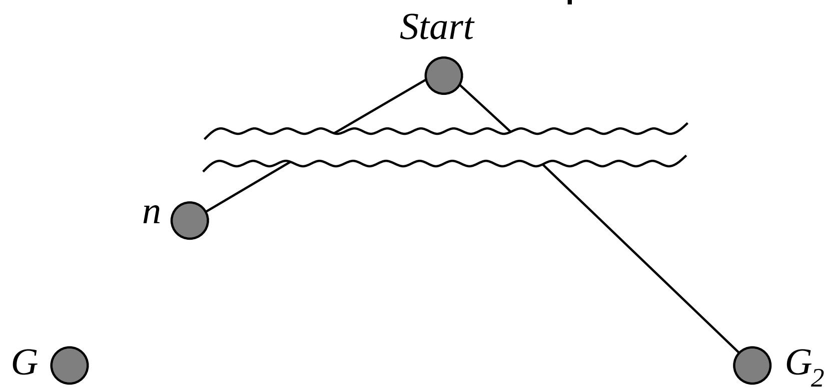


A* search example



Optimality of A* (standard proof)

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



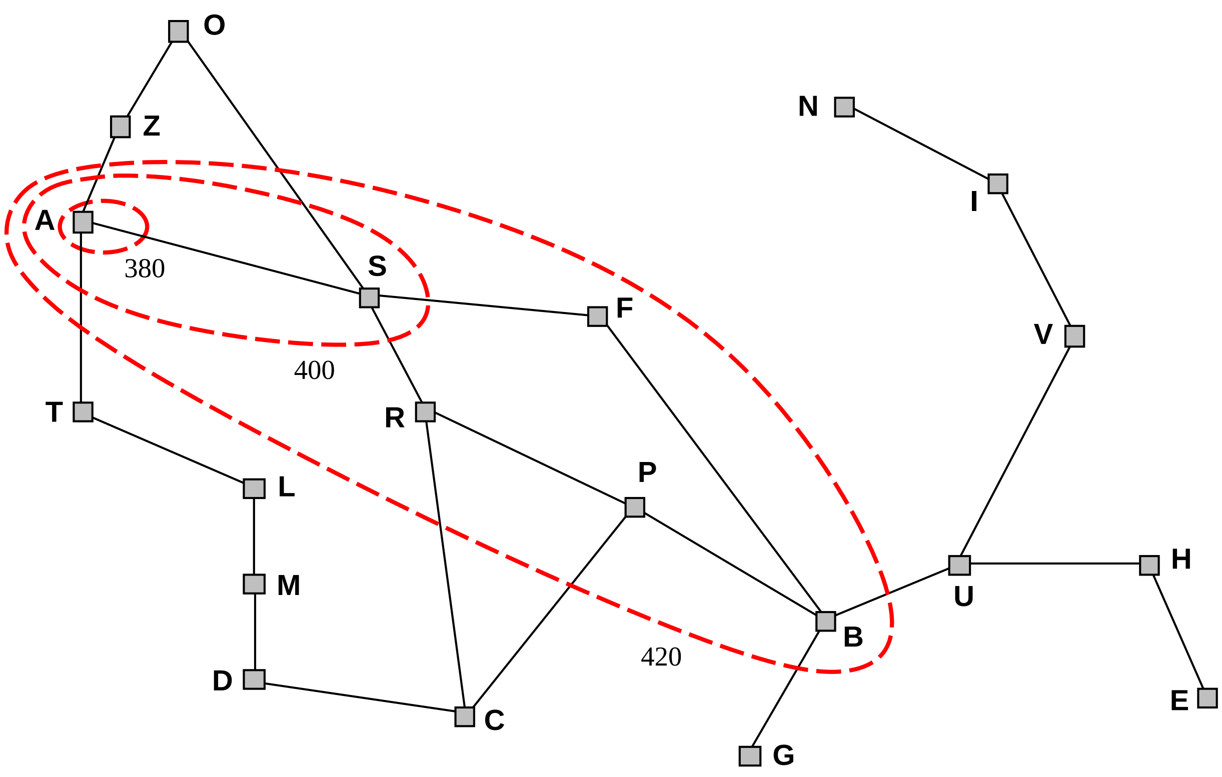
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

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

Optimality of A* (more useful)

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 $f_i < f_{i+1}$



Properties of A*

Complete??

Properties of A*

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

Time??

Properties of A*

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

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

Space??

Properties of A*

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

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

Space?? Keeps all nodes in memory

Optimal??

Properties of A*

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

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

Space?? Keeps all nodes in memory

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 $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

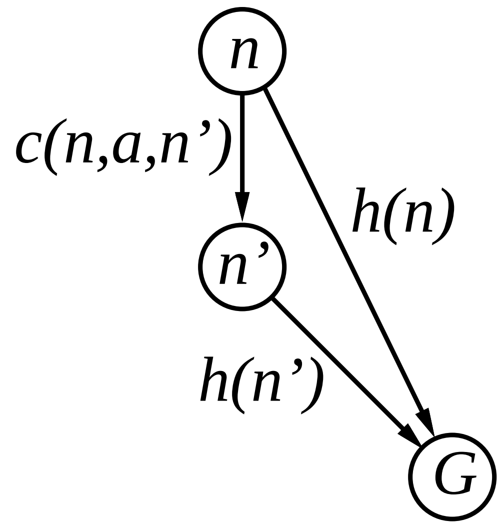
Proof of lemma: Consistency

A heuristic is consistent if

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

If h is consistent, we have

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



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

Exercise: Search Algs.

Consider the following scoring function for heuristic search:

$$f(n) = w \times g(n) + (1 - w) \times h(n) \quad \text{where} \quad 0 \le w \le 1$$

i. Which search algorithm do you get with *w* set to 0?

Exercise: Various Search Algs.

1) Prove that breadth-first search is a special case of uniform-cost search.

Exercise: Various Search Algs.

2) Prove that breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.

Exercise: Various Search Algs.

3) Prove that uniform-cost search is a special case of A^* search

Memory-Bounded Heuristic Search

- Since A* keeps all nodes in memory, it usually runs out of space before it runs out of time
- ➔ Memory-bounded heuristic search
  - Iterative-deepening A* (IDA*), where cutoff is the f-cost (g+h), rather than the depth.
  - Recursive best-first search – like best-first search, but only uses linear space
    - Keeps track of value of best alternative from any ancestor of current node
    - If current node exceeds this limit, then recursion unwinds back to alternative path

Memory-Bounded Heuristic Search

- Problem: IDA* and RBFS don’t use all the memory they could, leading to re-evaluation of states multiple times
- Memory-Bounded A* (MA*), and Simplified MA* (SMA*) are better.

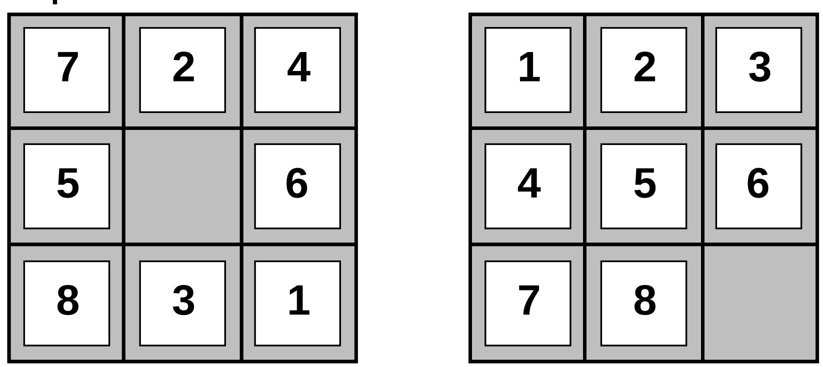
Simplified MA* (SMA*)

- Proceeds like A*, expanding best leaf until memory is full
- Then, it drops worst leaf node (i.e., one with highest f value)
- If all leaf nodes have same f value, then delete the oldest node
- SMA*
  - Is **complete** if *d* is less than memory size
  - Is **optimal** if any optimal solution is reachable
    - Otherwise, returns best reachable solution
- But, on hard problems, SMA* thrashes between many candidate solution paths
  - → Tradeoff between computation and memory

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)



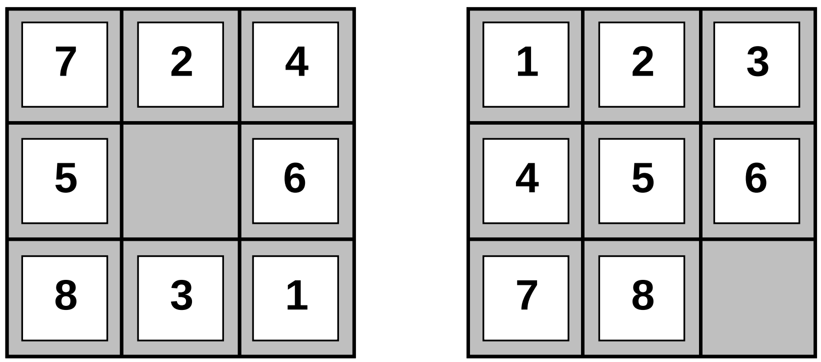
Start State **Goal State**

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

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)



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

Typical search costs:

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

Relaxed problems

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

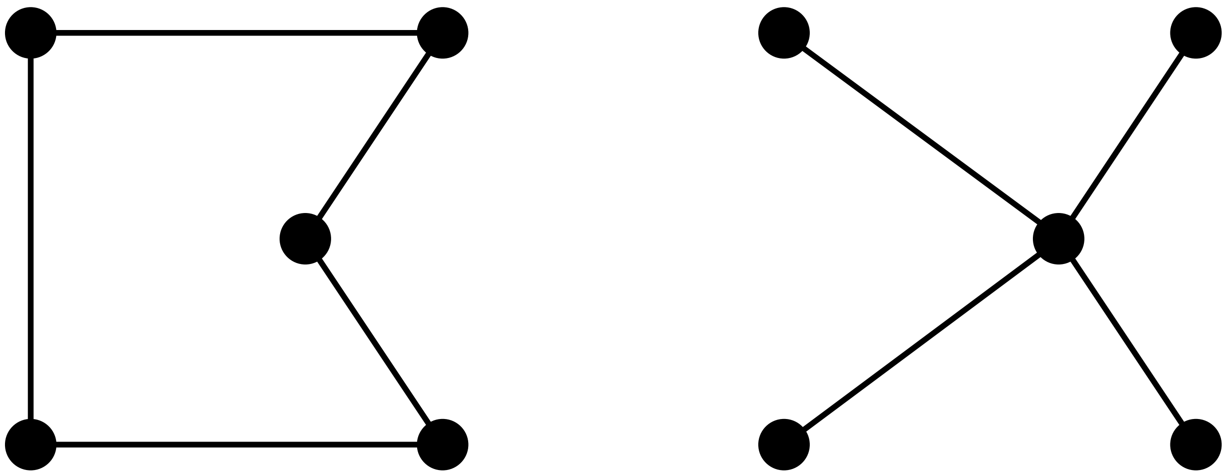
If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ gives the shortest solution

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

Relaxed problems contd.

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

Admissible Heuristics from Subproblems

- Can derive admissible heuristics from solution cost of subproblem of given problem
- Example: 8-puzzle: subproblem is solution to getting tiles 1, 2, 3, 4 in place (or any 4 of the tiles)
- Pattern databases: store exact solution costs for every possible subproblem instance (i.e., every configuration of the 4 tiles of the subproblem)
- Then compute admissible heuristic by looking up subproblem in database

Disjoint Pattern Databases

- If subproblems are independent, then can add costs of subproblems to create admissible heuristic
- E.g., for 8-puzzle, 2 subproblems: 1-2-3-4, 5-6-7-8.
  - Count cost of each subproblem only for the specified tiles (not all tiles)
  - Then, the two subproblem costs can be added

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
- also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems

Reading Assignment

- For Thursday: Chapter 4.3-4.5 (we’re skipping 4.1-4.2)
- For next week: Chapter 5