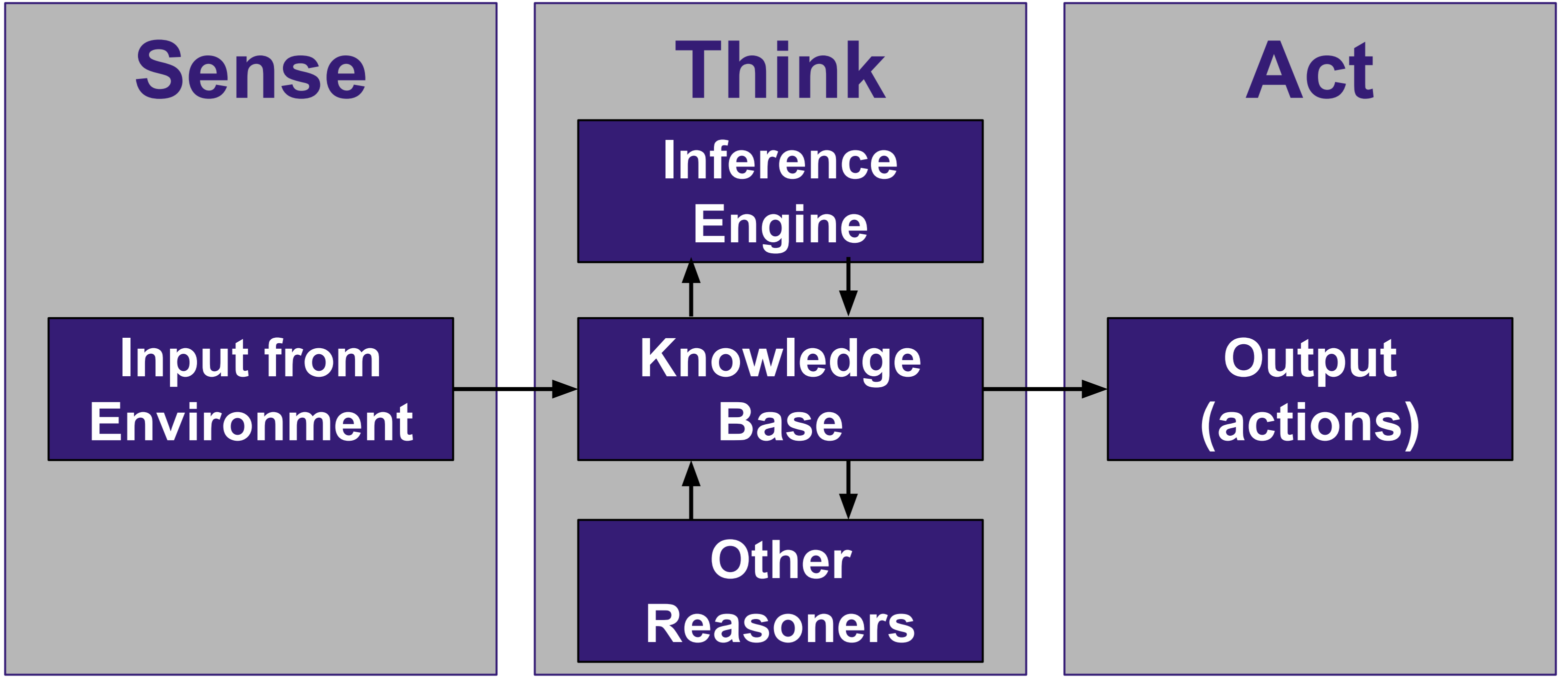
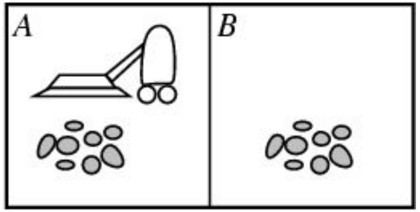
Informed Search

willie
(adapted from slides from Sara Owsley Sood)

Knowledge-based agent



Vacuum world knowledge-based agent using search



1. **Formulate problem and goal**
2. Search for a sequence of actions that will lead to the goal (the policy)
3. Execute the actions one at a time

<u>Well-defined problem:</u>

(State space)

Initial state

Goal test

Actions/Successor function

Path cost

Tree Search Algorithm

1. Add the initial state (root) to the <fringe>
2. **Choose a node (curr) to examine from the** <fringe>
   (if there is nothing in <fringe> - FAILURE)
3. Is curr a goal state?
   If so, SOLUTION
   If not, continue
4. Expand curr by applying all possible actions (add the new resulting states to the <fringe>)
5. Go to step 2

Search algorithm properties

b = ?? prob # of branches
d = shallowest depth
m = max depth of our tree

Time (using Big-O) aka: branching factor?

- approximate the number of nodes generated (not necessarily examined)

Space (using Big-O)

- the max # of nodes stored in memory at any time

Complete

- If a solution exists, will we find it?

Optimal

- If we return a solution, will it be the best/optimal (really just shallowest) solution

Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search

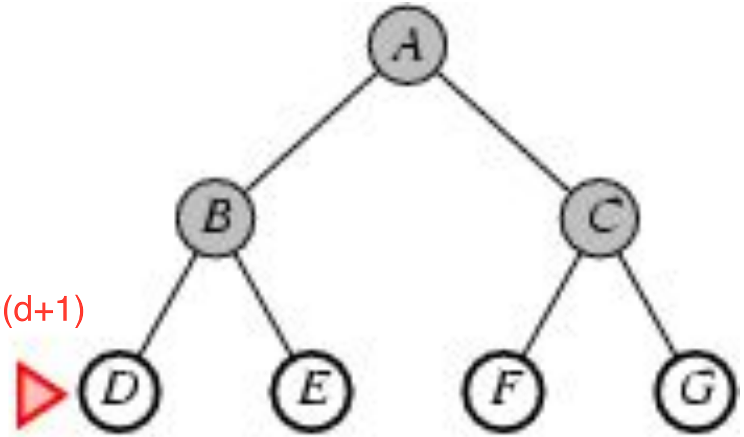
Breadth-first search

optimal, will do the shortest search

Expand shallowest unexpanded node

Implementation:

- *fringe* is a FIFO queue, i.e., new successors go at end



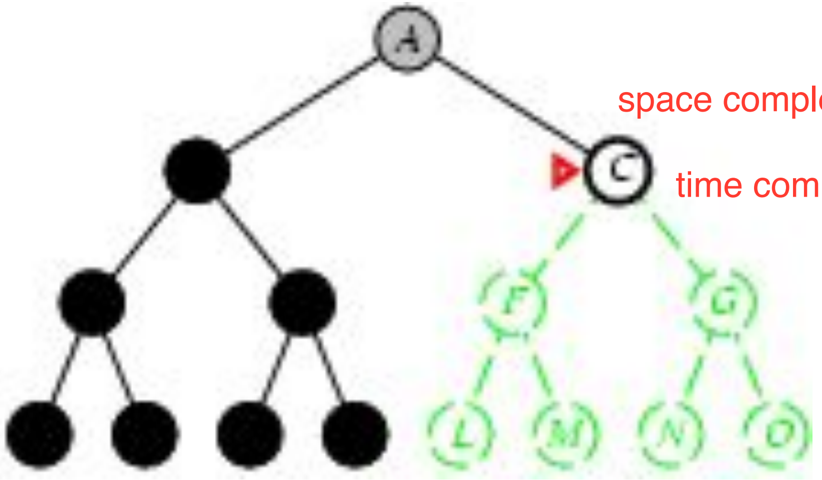
complexity will be ?? O(d+1)

Depth-first search

Expand deepest unexpanded node

Implementation:

- *fringe* is a LIFO queue, i.e., put successors at front



space complexity will be ?? O(b*m)

time complexity: ?? O(b^m + 1)

Depth-limited search

Depth-first search with depth limit *L*,

i.e., nodes at depth *L* have no successors

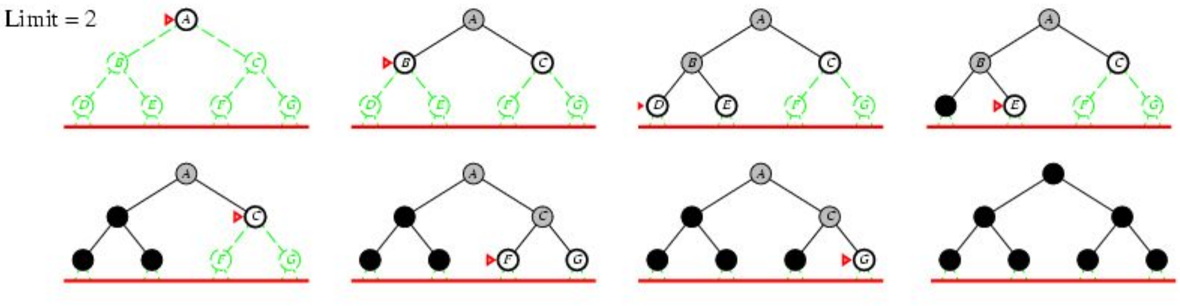
search to a certain level.
space complexity is O(b*L)
and time complexity is O(b^L)

Iterative deepening search L = 0

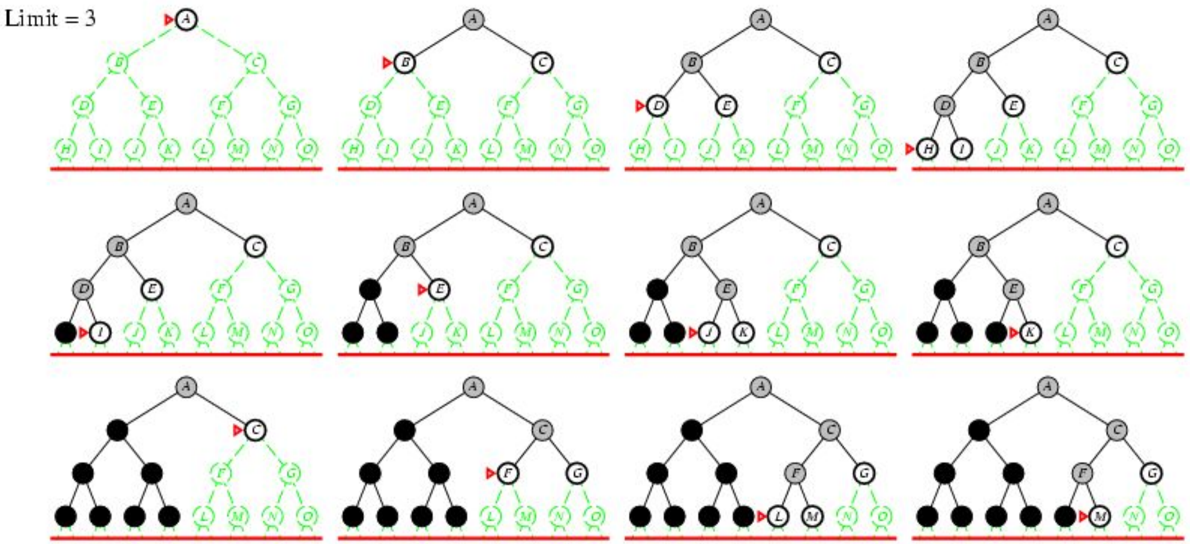
Limit = 0

Iterative deepening search L = 1

Iterative deepening search L = 2



Iterative deepening search L = 3



Properties of iterative deepening search

Space

- $O(bd)$

Complete?

if there is a solution, it will find it

- Yes

Optimal?

only increasing level 1 by 1, we will find the optimal method

- Yes

Iterative deepening search

Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

Time?

$L = 0$: 1

$L = 1$: $1 + b$

$L = 2$: $1 + b + b^2$

$L = 3$: $1 + b + b^2 + b^3$

…

$L = d$: $1 + b + b^2 + b^3 + \ldots + b^d$

Overall:

$$(d+1)b^0 + (d)b^1 + (d-1)b^2 + (d-2)b^3 + \ldots + 2b^{d-1} + b^d$$

$$= O(b^d)$$

asymptotically it’s more or less the same effort (b^d)

the cost of the repeat of the lower levels is subsumed by the cost at the highest level

Properties of iterative deepening search

Time?

$(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?

$O(bd)$

Optimal?

Yes

Complete?

Yes

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Tree Search Algorithm

1. Add the initial state (root) to the <fringe>
2. **Choose a node (curr) to examine from the** <fringe>

    (if there is nothing in <fringe> - FAILURE)
3. Is curr a goal state?

    If so, SOLUTION

    If not, continue
4. Expand curr by applying all possible actions (add the new resulting states to the <fringe>)
5. Go to step 2

Search trees for 8-puzzle

2	8	3
1	6	4
7		5

Initial state

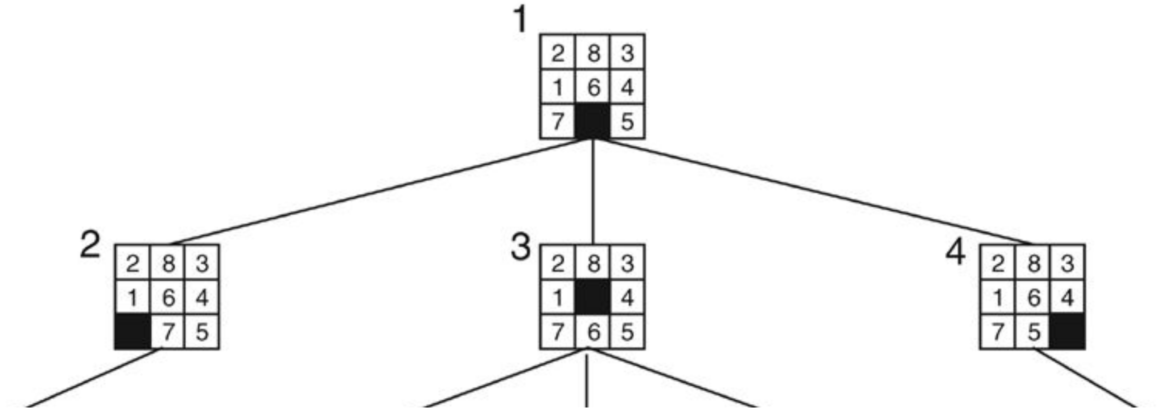
1	2	3
8		4
7	6	5

Goal

What does the search tree using BFS look like for this problem?

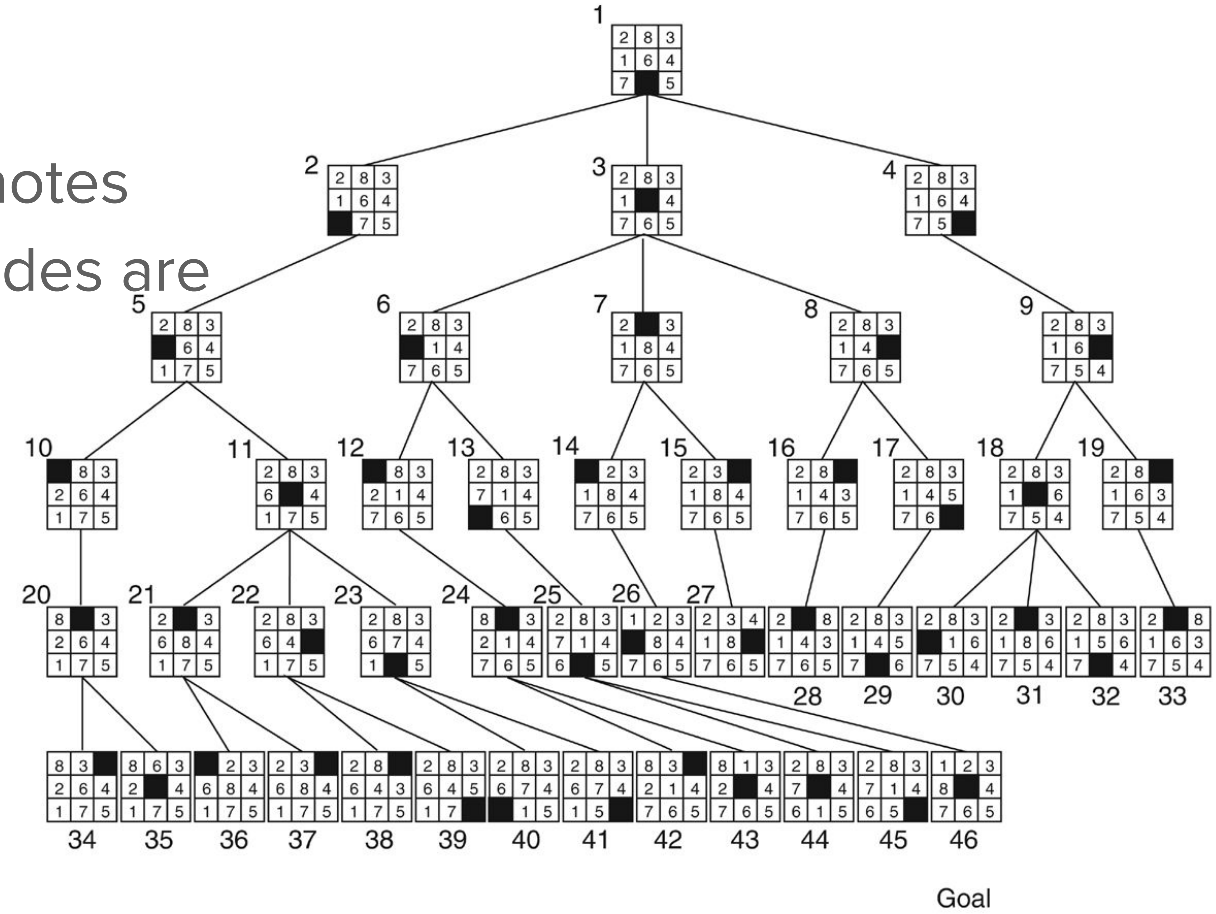
What does the DFS search tree look like?

Start of BFS search tree



BFS

Number on node denotes order in which the nodes are visited.

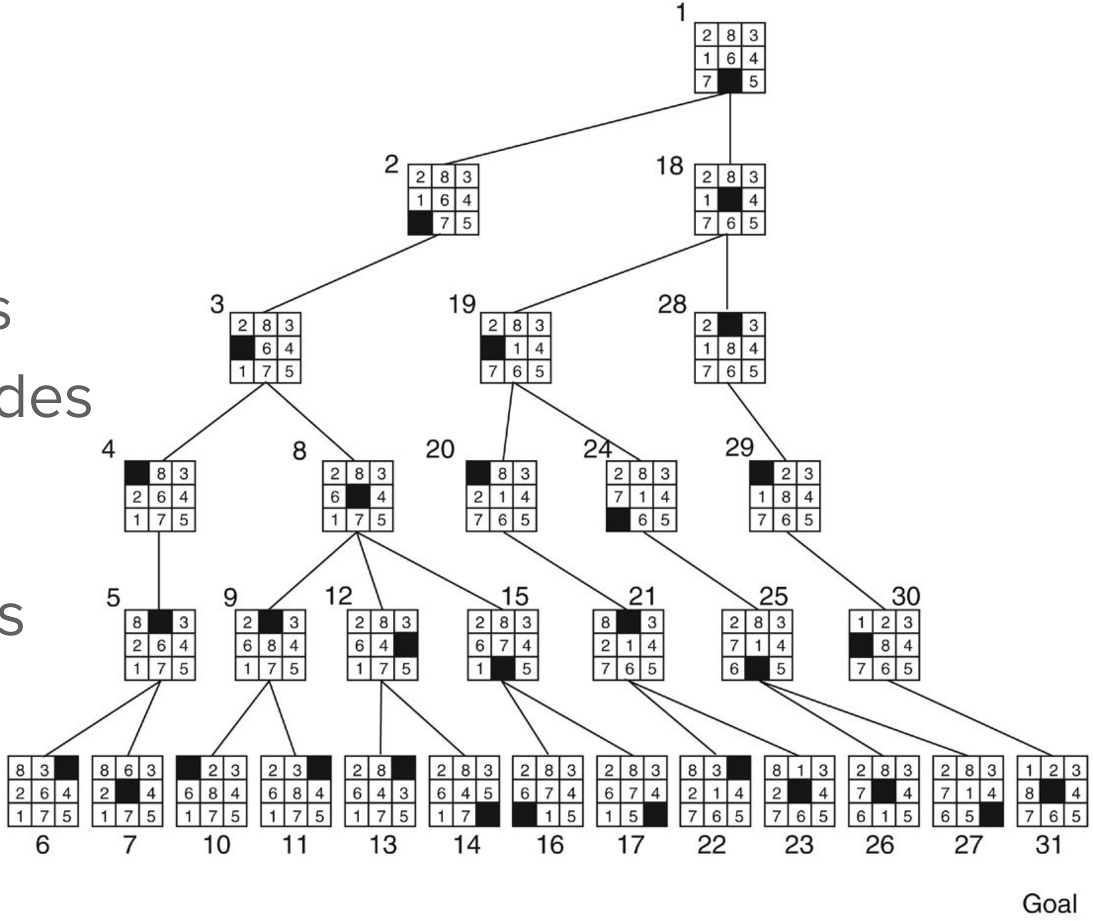


DFS

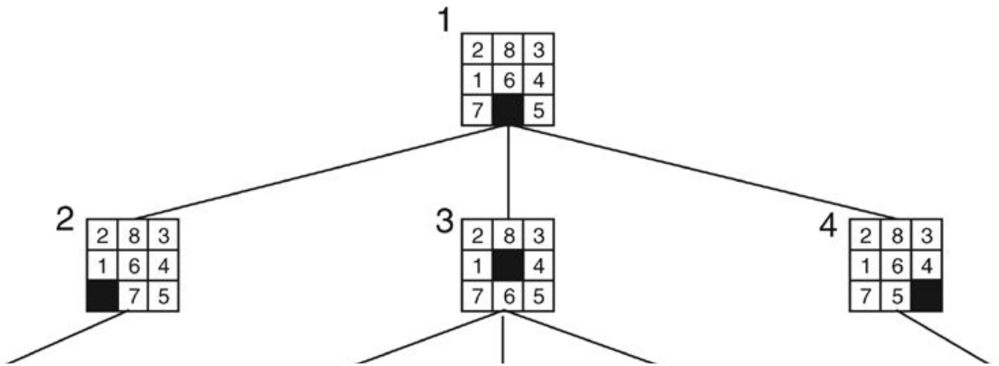
Depth bound of 5

Number on node denotes the order in which the nodes are examined

Pic is missing some nodes that were added but not examined



Can we do better?



Do we know which of the first choices is best?

Can we make an intelligent choice?

Two heuristics applied to states in the 8-puzzle.

<table>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td></td><td>7</td><td>5</td></tr></table></td><td>5</td><td>6</td></tr>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td><td>3</td><td>4</td></tr>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td>7</td><td>5</td><td></td></tr></table></td><td>5</td><td>6</td></tr>
<tr><td></td><td>Tiles out of place</td><td>Sum of distances out of place</td></tr>
</table>

1	2	3
8		4
7	6	5

Goal

Best-first search

Idea: use an **evaluation function** *f(n)* for each node

- estimate of "desirability"
- Expand most desirable unexpanded node

Implementation:

Order the nodes in fringe in decreasing order of desirability

Special cases:

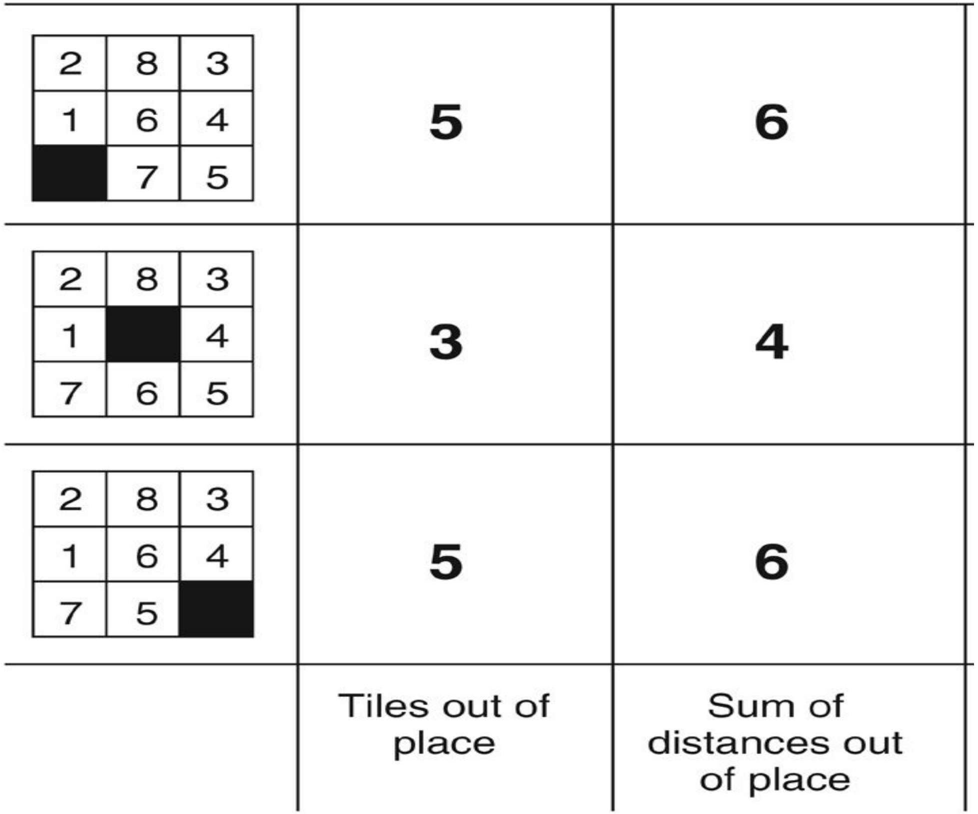
- greedy best-first search
- A* search

Greedy best-first search

Evaluation function

f(n) = h(n) (**h**euristic)

Estimate of cost from
n to *goal*



A* search

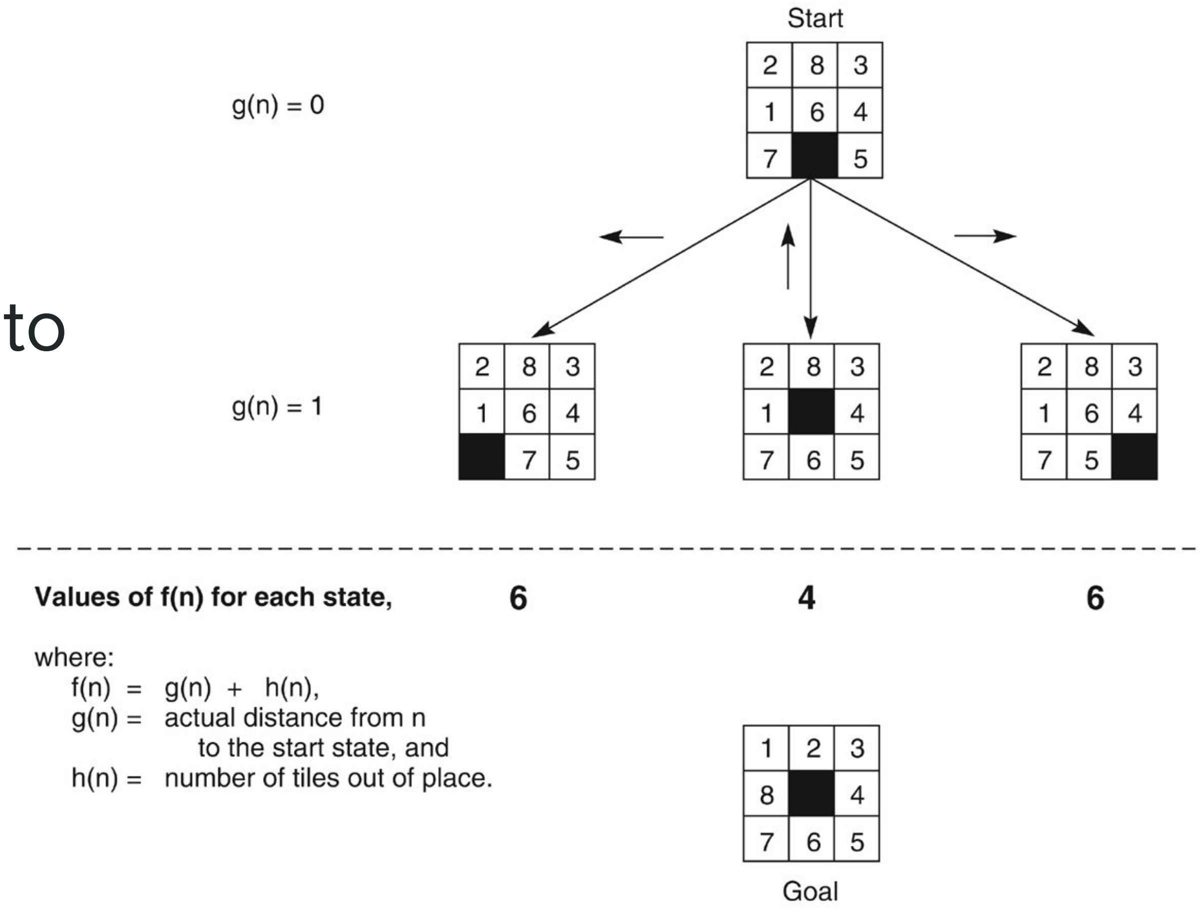
Idea: avoid expanding paths that are already expensive

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

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

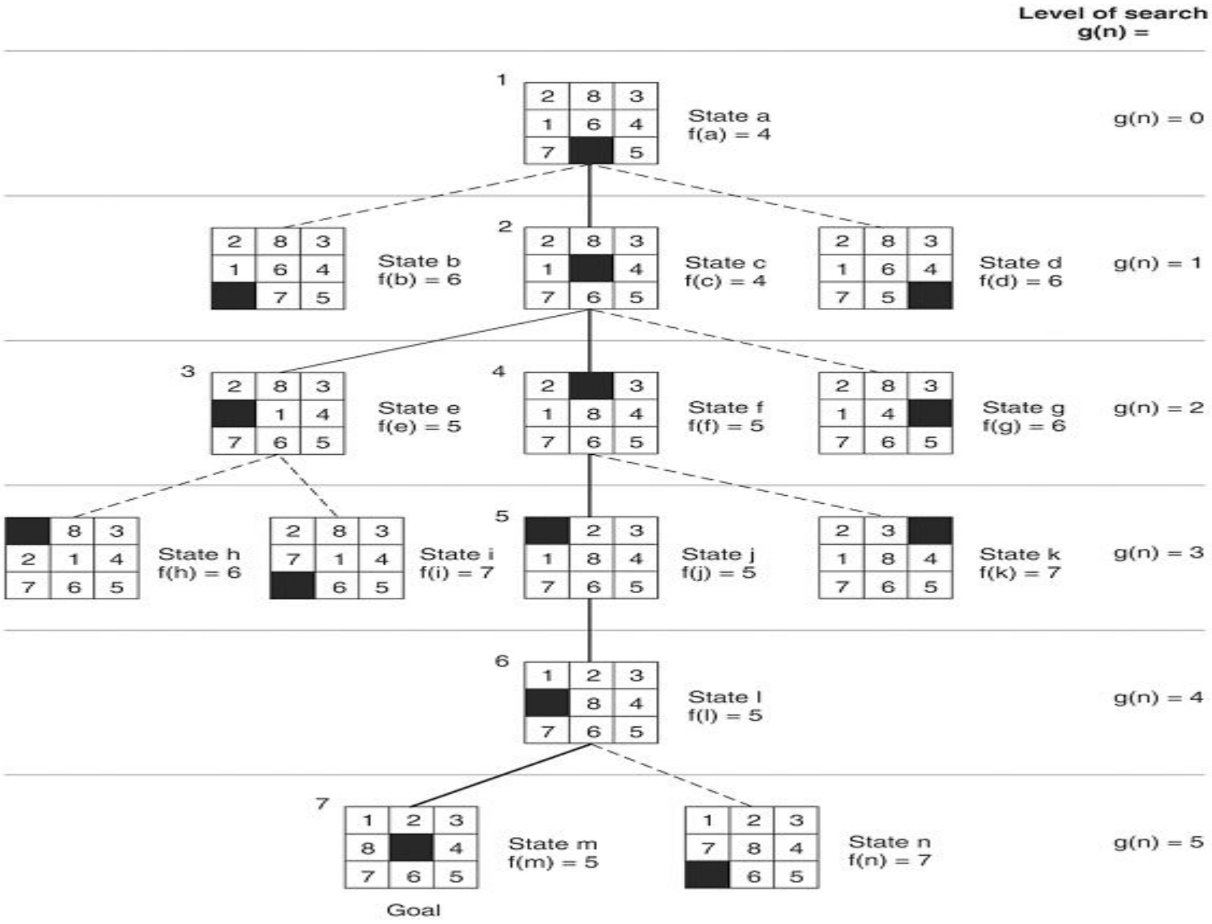
A* in 8-puzzle

The evaluation function f applied to states in the 8-puzzle

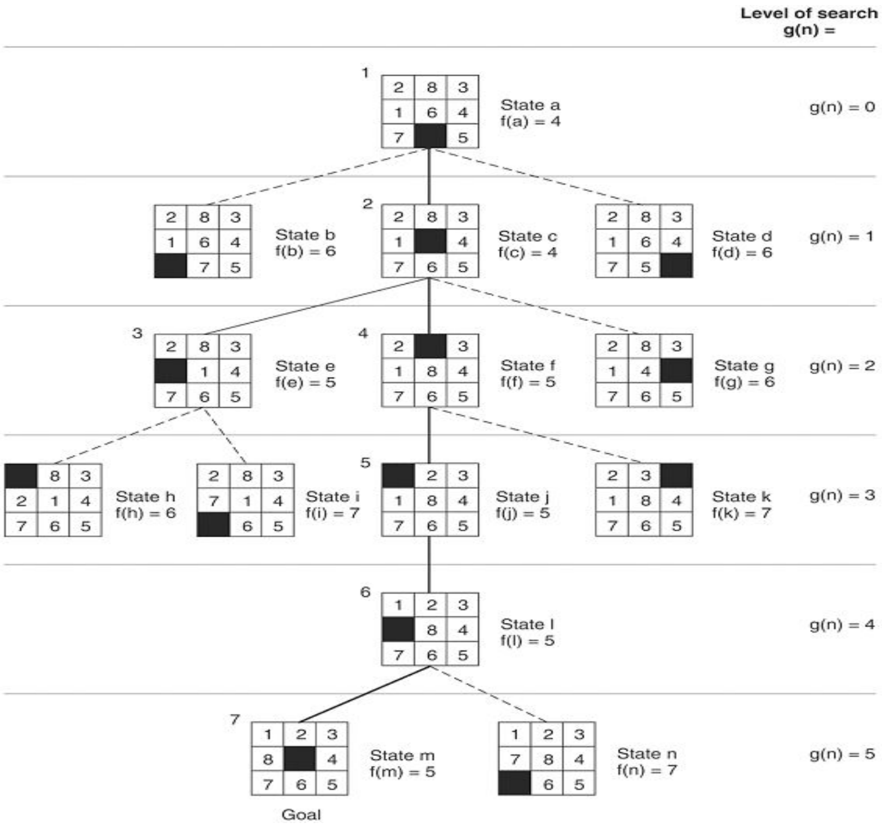


State space

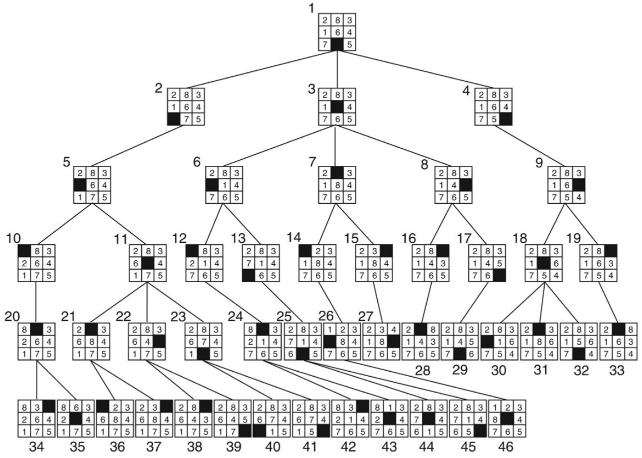
Heuristic search fo 8-puzzle is much more efficient than BFS and DFS

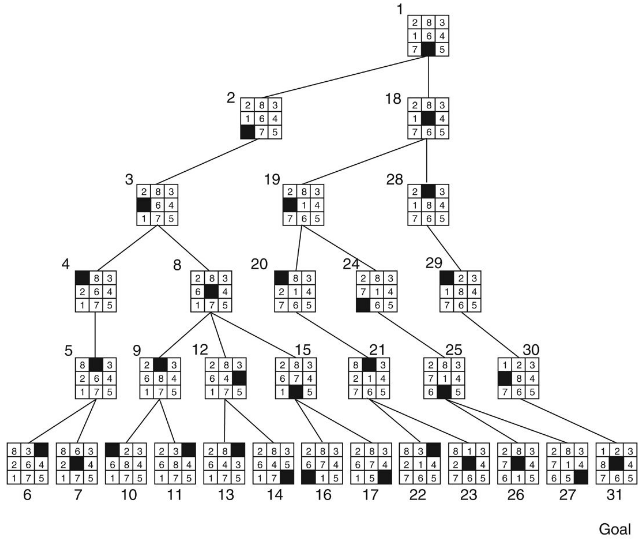


Compare



Level of search
g(n) =
State a
f(a) = 4
g(n) = 0
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
State e
f(e) = 5
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
State l
f(l) = 5
g(n) = 4
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal



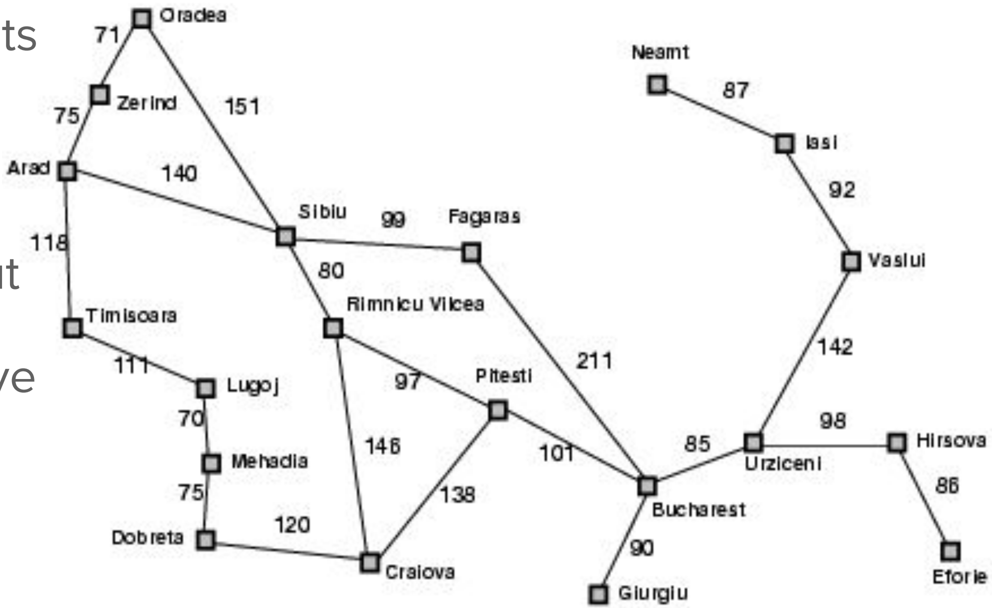


Goal

Intelligent order of Expansion?

TreeSearch (and GraphSearch) expands its nodes in a given order

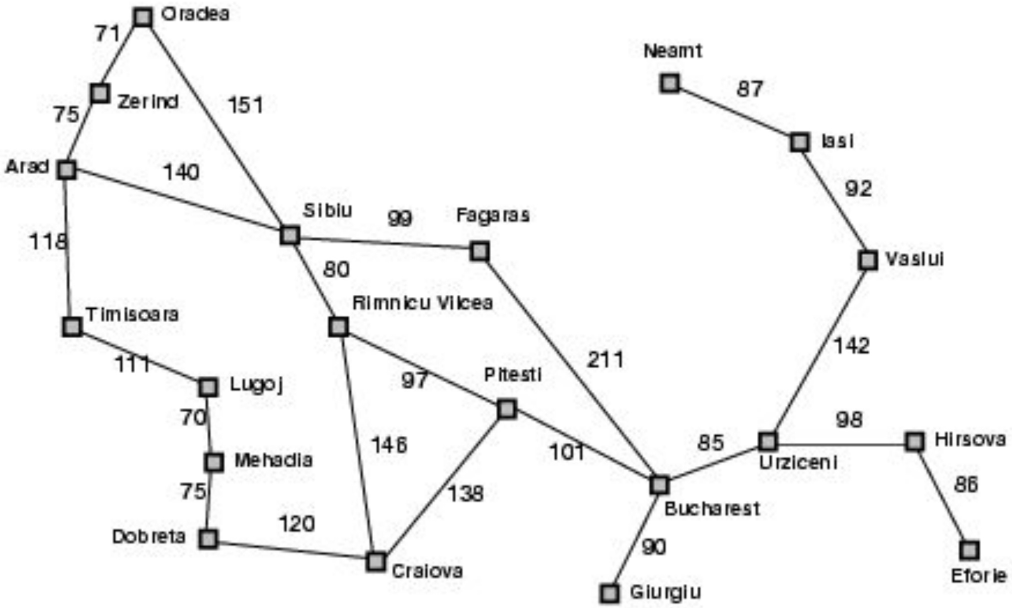
Can we be "smart" about the order in which we expand nodes to improve the search?



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

Romania with step costs in km



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

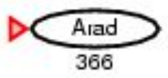
Greedy best-first search

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

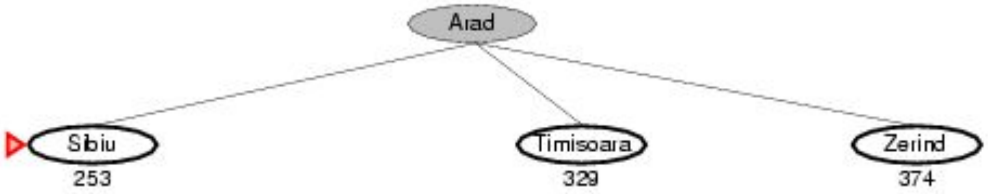
estimate of cost from *n* to *goal*

$h_{SLD}(n)$ = straight-line distance from n to Bucharest

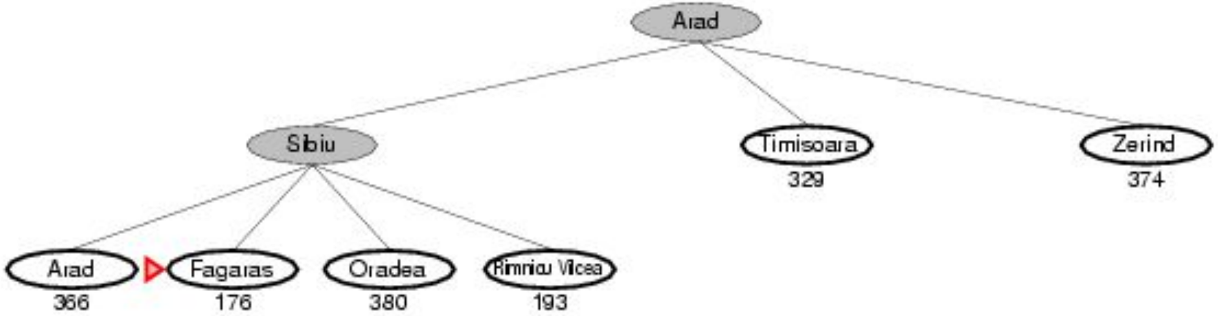
Greedy best-first search example



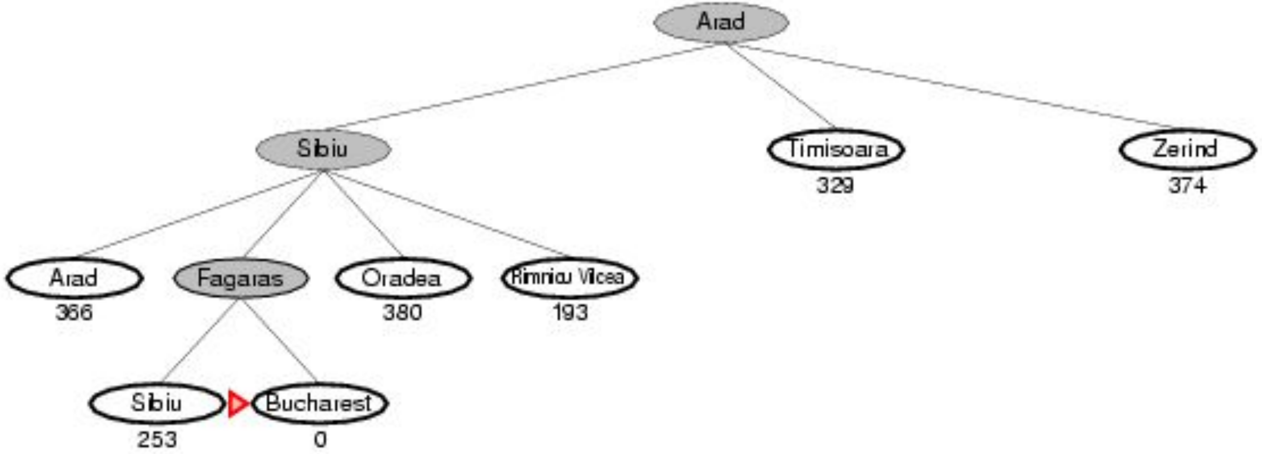
Greedy best-first search example



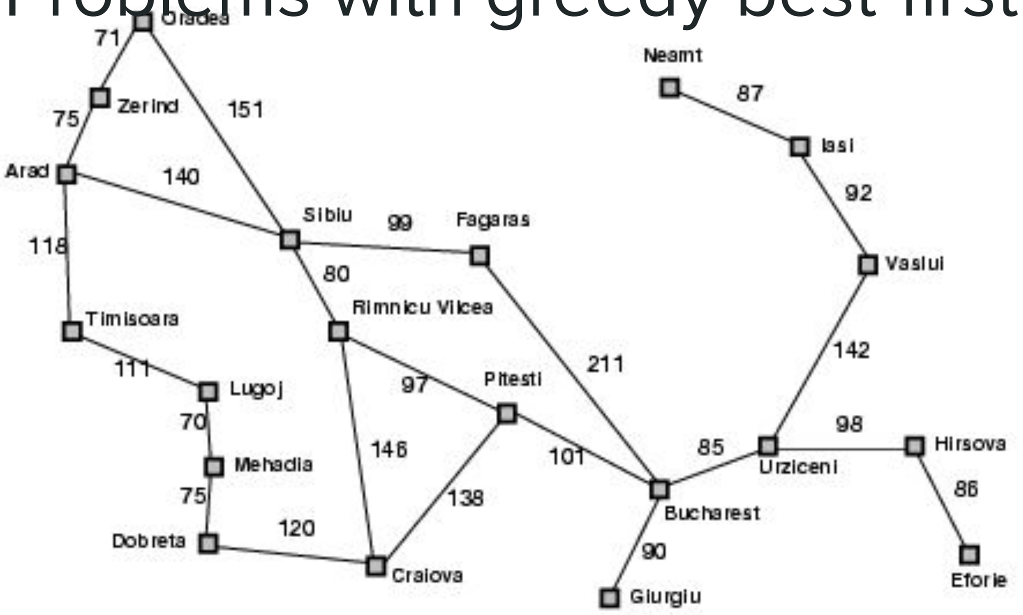
Greedy best-first search example



Greedy best-first search example



Problems with greedy best-first search?



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

SLD to Fagaras

Neamt – 180

Iasi – 200

Vasliu – 220

Fagaras - 0

A* search

Idea: avoid expanding paths that are already expensive

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

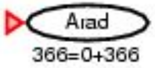
$g(n)$ = cost so far to reach n

$h(n)$ = estimated cost from n to goal

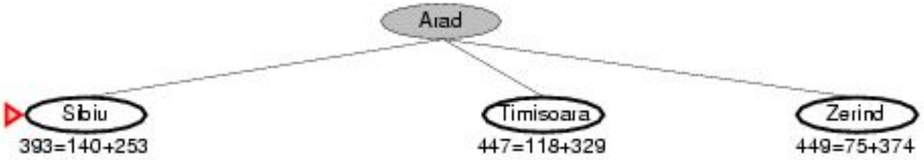
$f(n)$ = estimated total cost of path through n to goal

(the evaluation of the desirability of n)

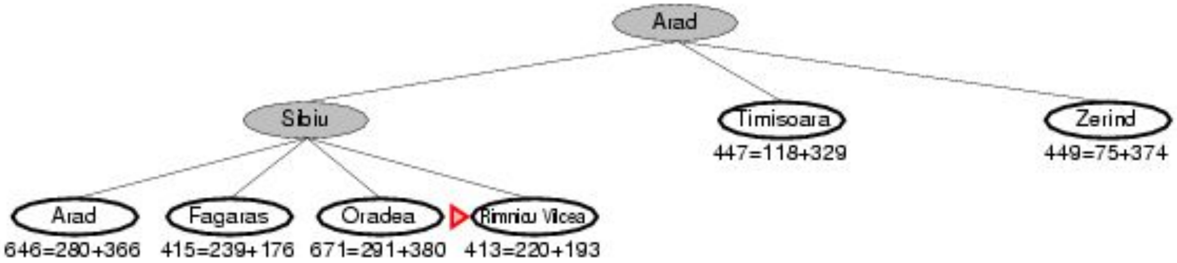
A* search example



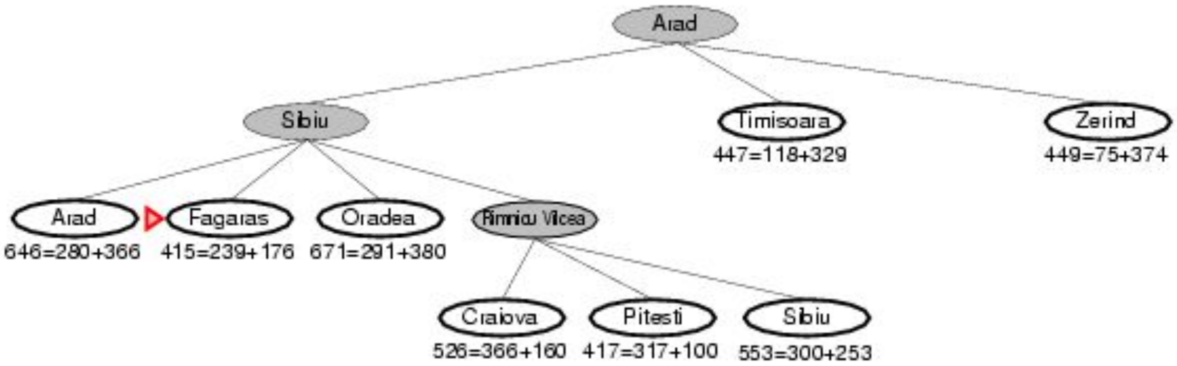
A* search example



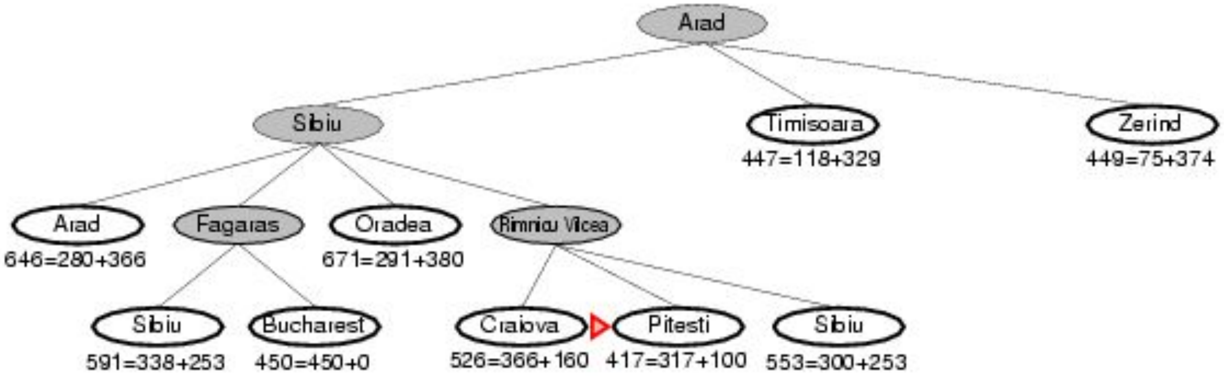
A* search example



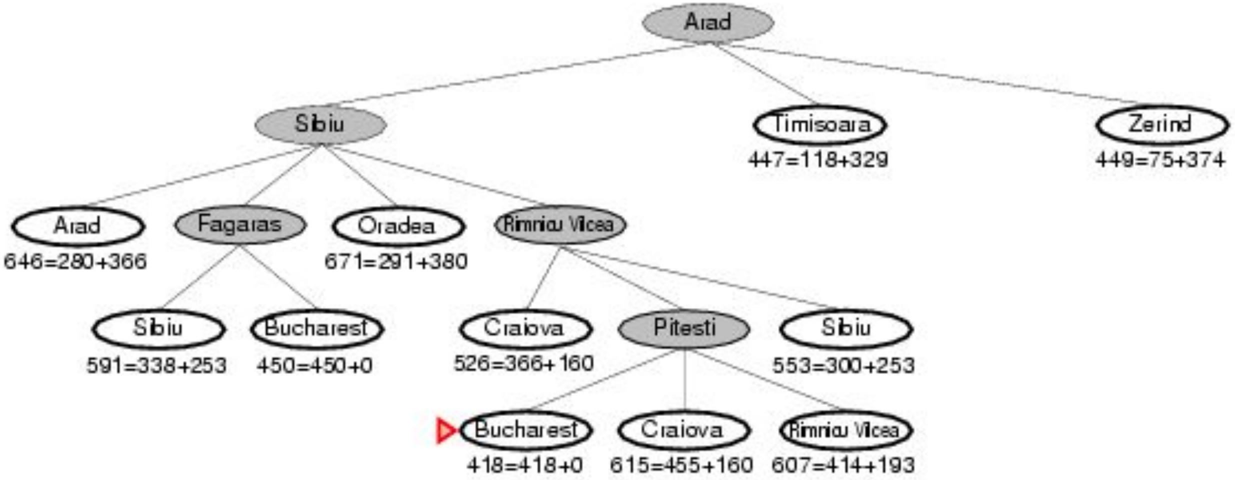
A* search example



A* search example



A* search example



shortest path b/c of the heuristic
straight line is always nice and admissible heuristic (also consistent)

Admissible heuristics

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.

An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**

Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

TreeSearch algorithm becomes GraphSearch

1. start with the initial node as *curr*
2. have I been to *curr* before? (is it in CLOSED)
3. is *curr* the goal?
4. if neither, expand *curr* - add children/successors to OPEN, add curr to CLOSED
5. choose a new node *curr* according to the smallest f(n) & go to step 2