بسم اللّه الرحمن الرحیم

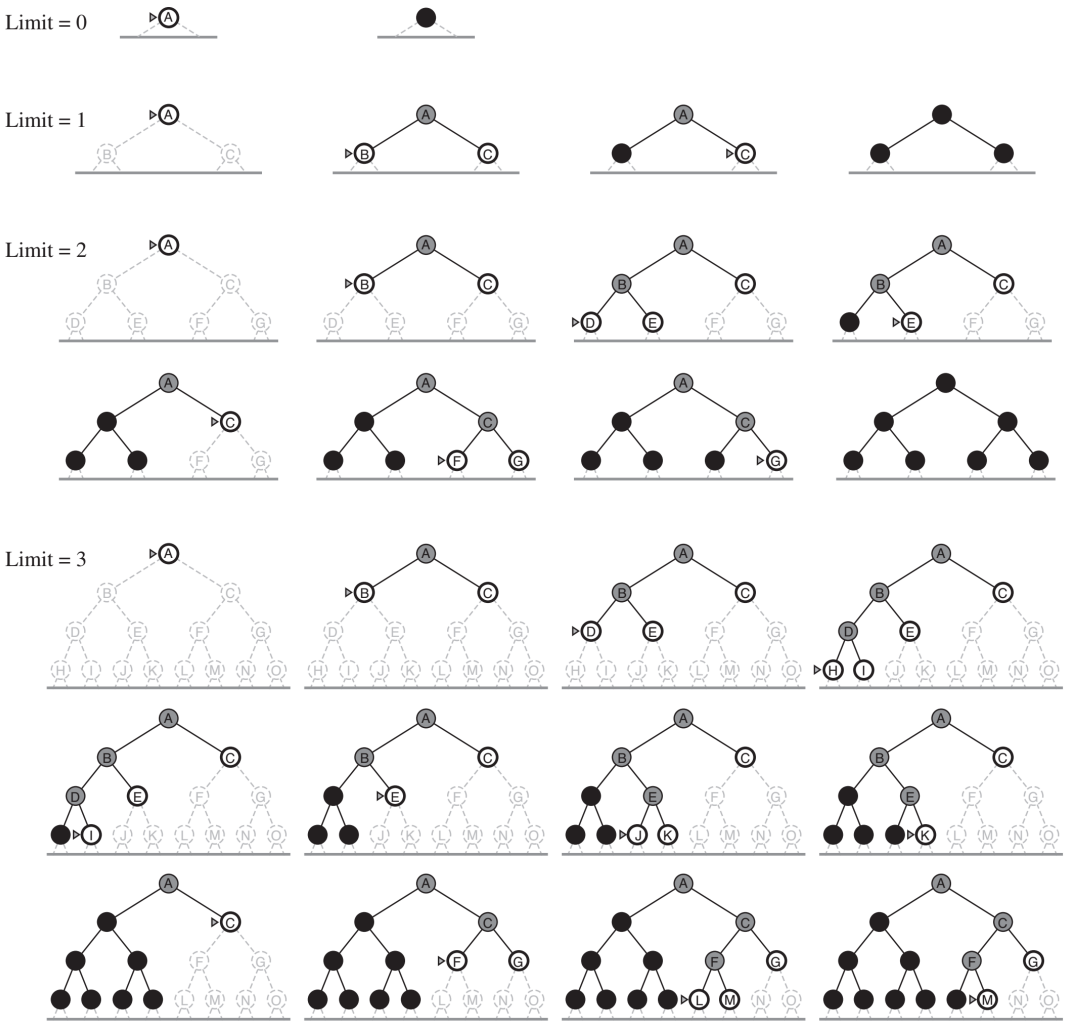
Systematic Search Algorithms (Chapter 3)

Iterative deepening depth-first search

Gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found. This will occur when the depth limit reaches d, the depth of the shallowest goal node.

Iterative deepening combines the benefits of depth-first and breadth-first search. Like depth-first search, its memory requirements are modest: $O(bd)$ to be precise. Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node.

IDS



IDS

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure

  for *depth* = 0 **to** ∞ **do**

    result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)

    if *result* $\neq$ cutoff **then return** *result*

IDS may seem wasteful because states are generated multiple times. It turns out this is not too costly. The reason is that in a search tree with the same (or nearly the same) branching factor at each level, most of the nodes are in the bottom level, so it does not matter much that the upper levels are generated multiple times. In an iterative deepening search, the nodes on the bottom level (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which are generated d times. So the total number of nodes generated in the worst case is

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \cdots + (1)b^d,$$

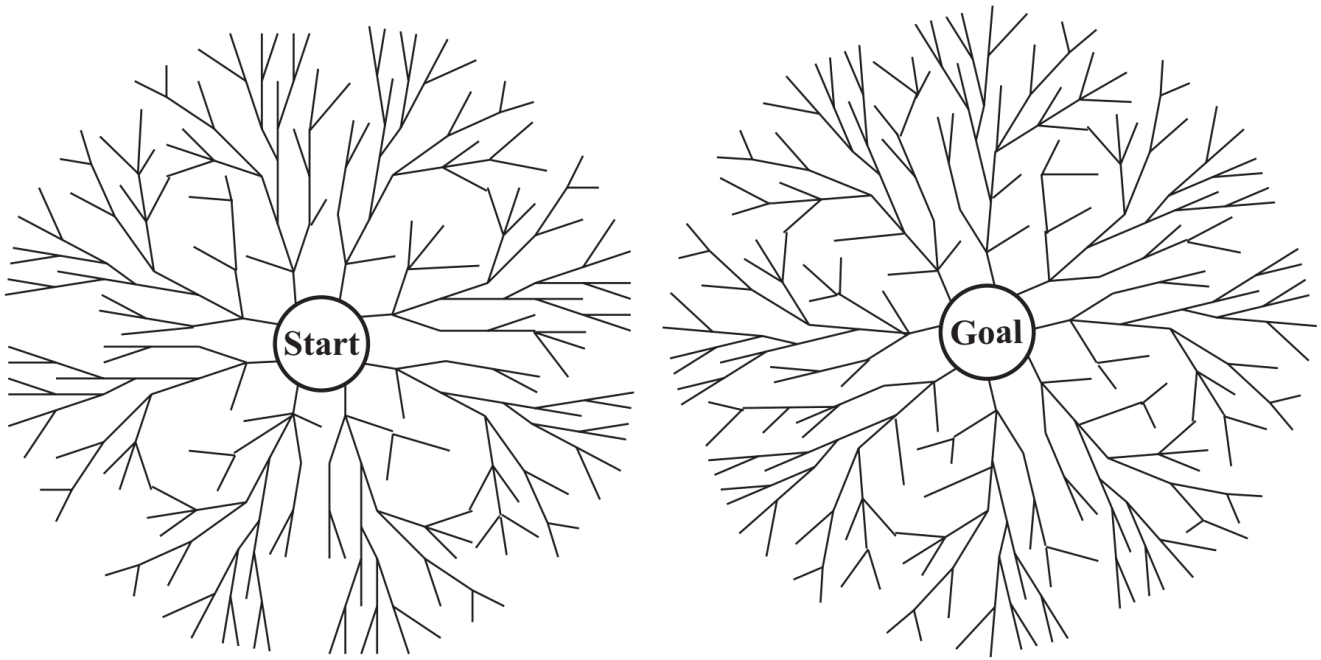
which gives a time complexity of $O(b^d)$—asymptotically the same as breadth-first search. There is some extra cost for generating the upper levels multiple times, but it is not large.

For example, if $b = 10$ and $d = 5$, the numbers are

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110$$

Bidirectional Search



The idea behind bidirectional search is to run two simultaneous searches—one forward from the initial state and the other backward from the goal—hoping that the two searches meet in the middle.

The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d,or in the figure, the area of the two small circles is less than the area of one big circle centered on the start and reaching to the goal.

Bidirectional search is implemented by replacing the goal test with a check to see whether the frontiers of the two searches intersect; if they do, a solution has been found.

The first such solution found may not be optimal, even if the two searches are both breadth-first; some additional search is required to make sure there isn't another short-cut across the gap.

روش جستجوی دوجهته برای هر مسئله‌ای قابل اعمال نیست. مثلاً برای مسئلهٔ **route-finding** و مسئلهٔ **8-puzzle** این روش قابل اعمال است؛ اما برای n- **queens** خیر.

Example: If a problem has solution depth $d = 6$, and each direction runs breadth-first search one node at a time, then in the worst case the two searches meet when they have generated all of the nodes at depth 3. For $b = 10$, this means a total of 2,220 node generations, compared with 1,111,110 for a standard breadth-first search.

The time complexity of bidirectional search using breadth-first searches in both directions is $O(b^{d/2})$. The space complexity is also $O(b^{d/2})$.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Informed (Heuristic) Search Strategies

We use problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy. (In many cases finds a solution faster than uninformed search.)

The general approach we consider is called **best-first search**. Best-first search is an instance of the general Tree-Search or Graph-Search algorithm in which a node is selected for expansion based on an **evaluation function**, $f(n)$.

The evaluation function is constructed as a cost estimate, so the node with the lowest evaluation is expanded first.

We use f to order the priority queue. The choice of f determines the search strategy.

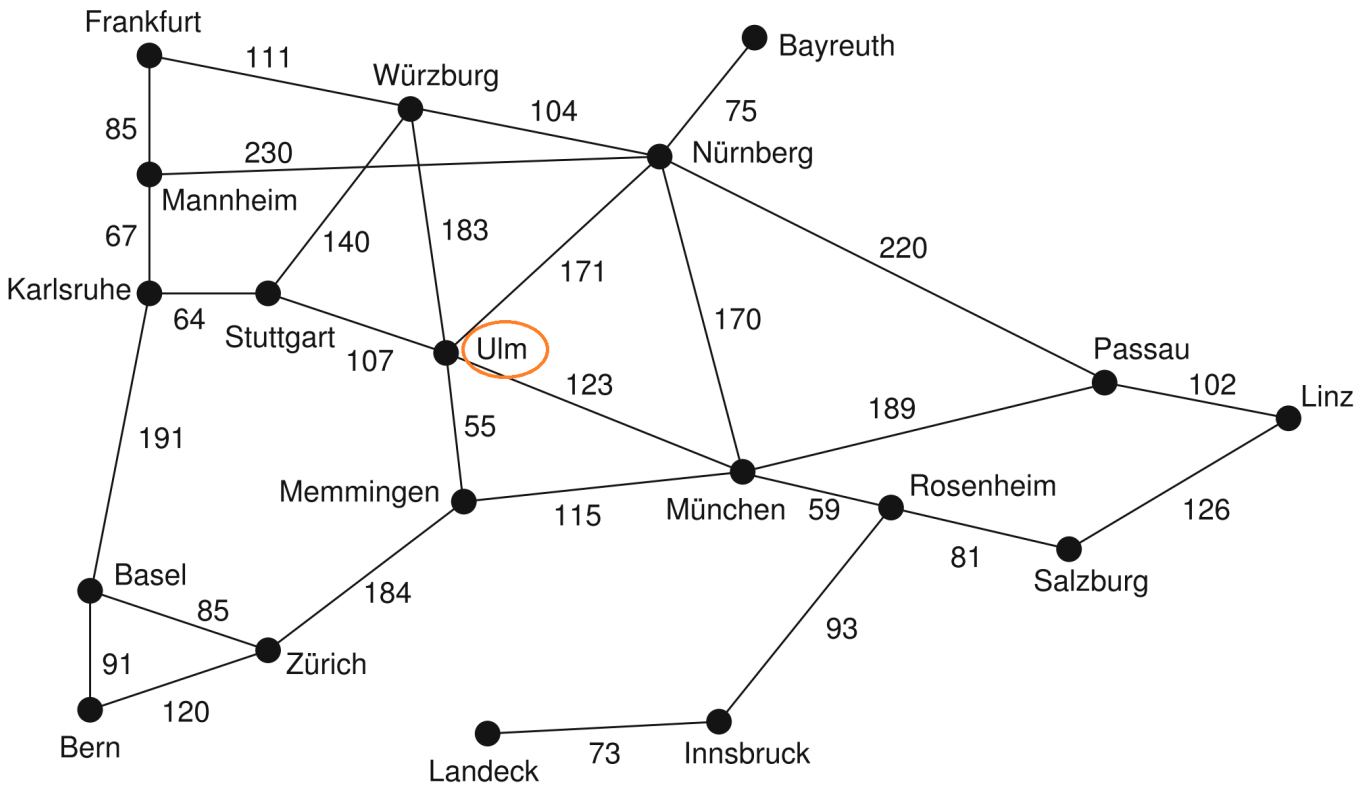
Most best-first algorithms include as a component of f a heuristic function, denoted $h(n)$:
$h(n) =$ estimated cost of the cheapest path from the state at node n to a goal state.
Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.

Greedy best-first search (Greedy Search)

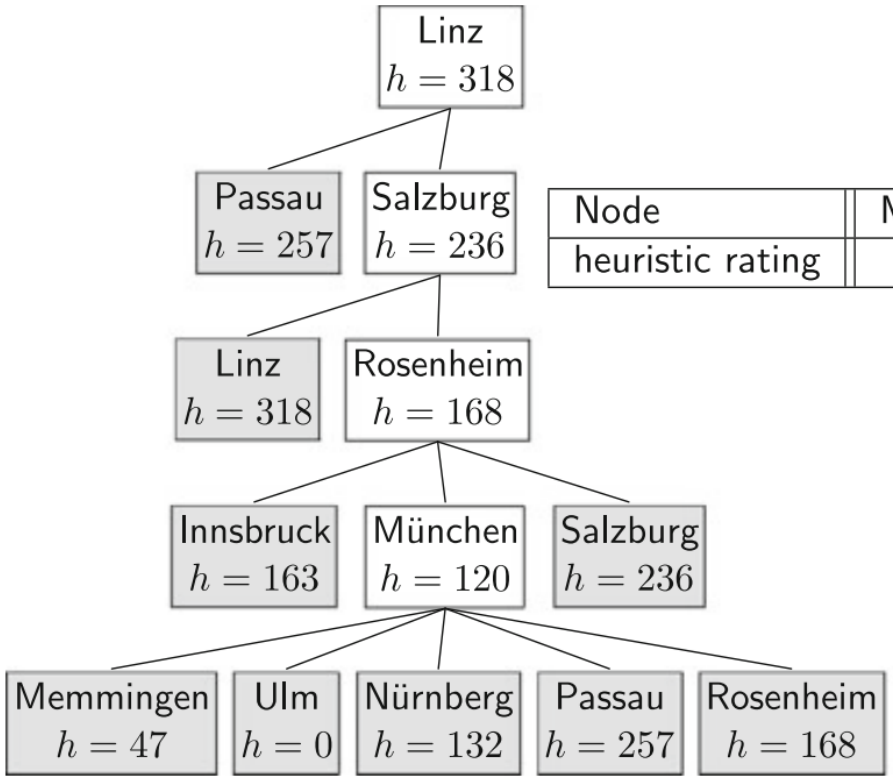
Tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$**.**

Example: route-finding (trip-planning) problem

$h(n) =$ **the straight-line distance (flying distance) heuristic**

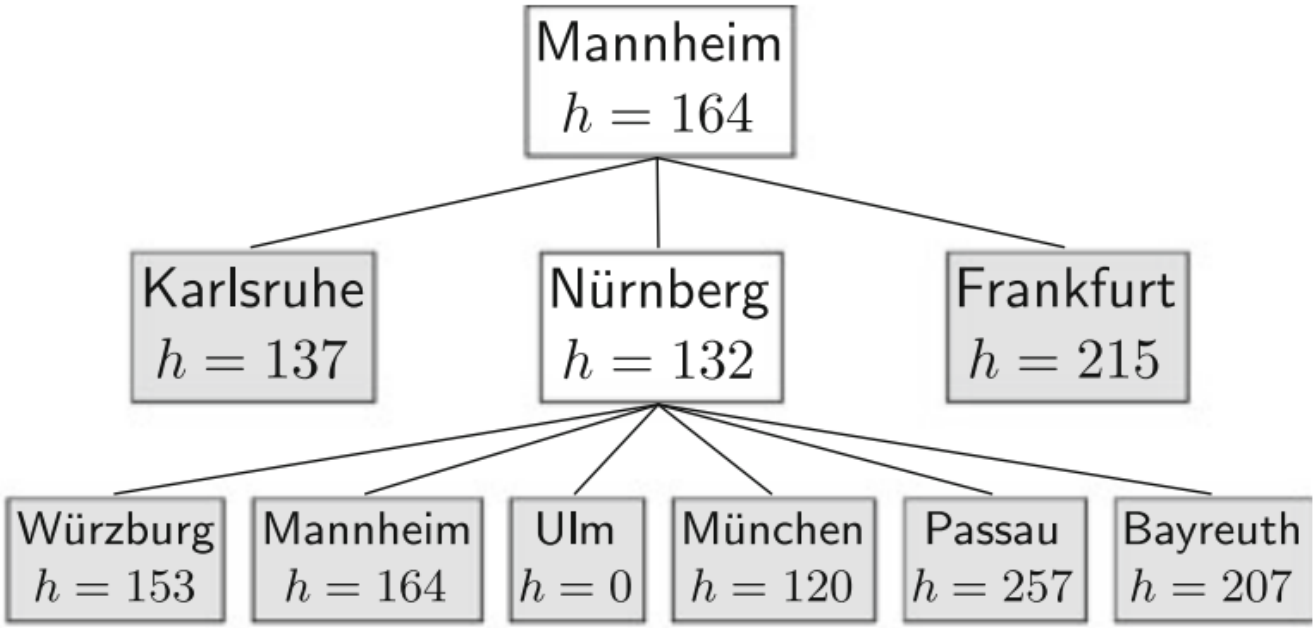


Basel	204
Bayreuth	207
Bern	247
Frankfurt	215
Innsbruck	163
Karlsruhe	137
Landeck	143
Linz	318
München	120
Mannheim	164
Memmingen	47
Nürnberg	132
Passau	257
Rosenheim	168
Stuttgart	75
Salzburg	236
Würzburg	153
Zürich	157



Node	München	Innsbruck	Salzburg	Passau	Linz
heuristic rating	120	163	236	257	318

Unfortunately, this search does not always find the optimal solution.

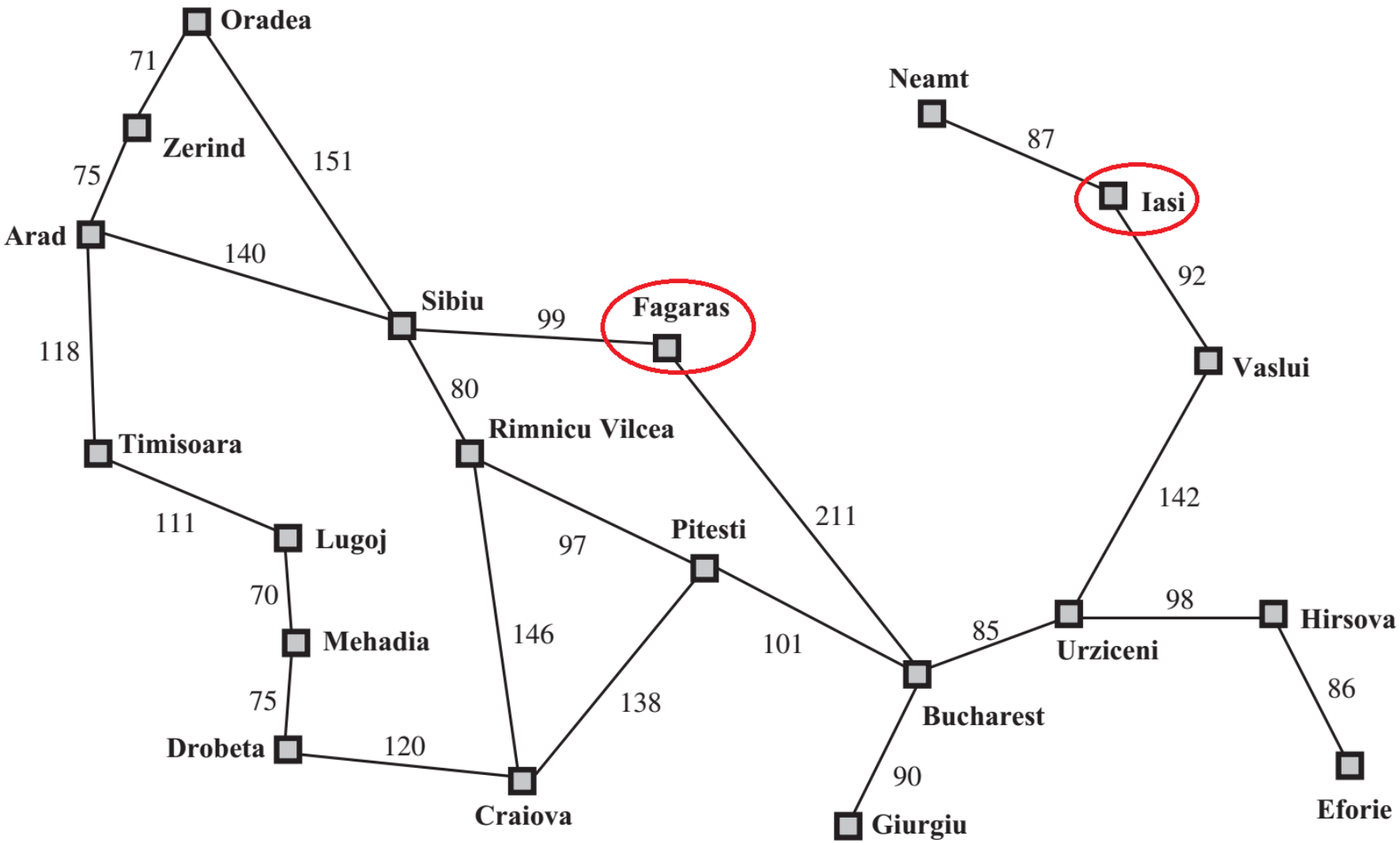


Node	München	Innsbruck	Salzburg	Passau	Linz
heuristic rating	120	163	236	257	318

The Mannheim–Nürnberg–Ulm path has a length of 401 km. The route Mannheim–Karlsruhe–Stuttgart–Ulm would be significantly shorter at 238 km.

Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.

Consider the problem of getting from Iasi to Fagaras. The heuristic suggests that Neamt be expanded first because it is closest to Fagaras, but it is a dead end. The solution is to go first to Vaslui—a step that is actually farther from the goal according to the heuristic—and then to continue to Urziceni, Bucharest, and Fagaras. The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier, Iasi is closer to Fagaras than Vaslui is, and so Iasi will be expanded again, leading to an infinite loop.


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
Craiova
90
Giurgiu
Eforie
Neamt
87
Iasi
92
Vaslui
142

The graph search version is complete in finite spaces, but not in infinite ones.

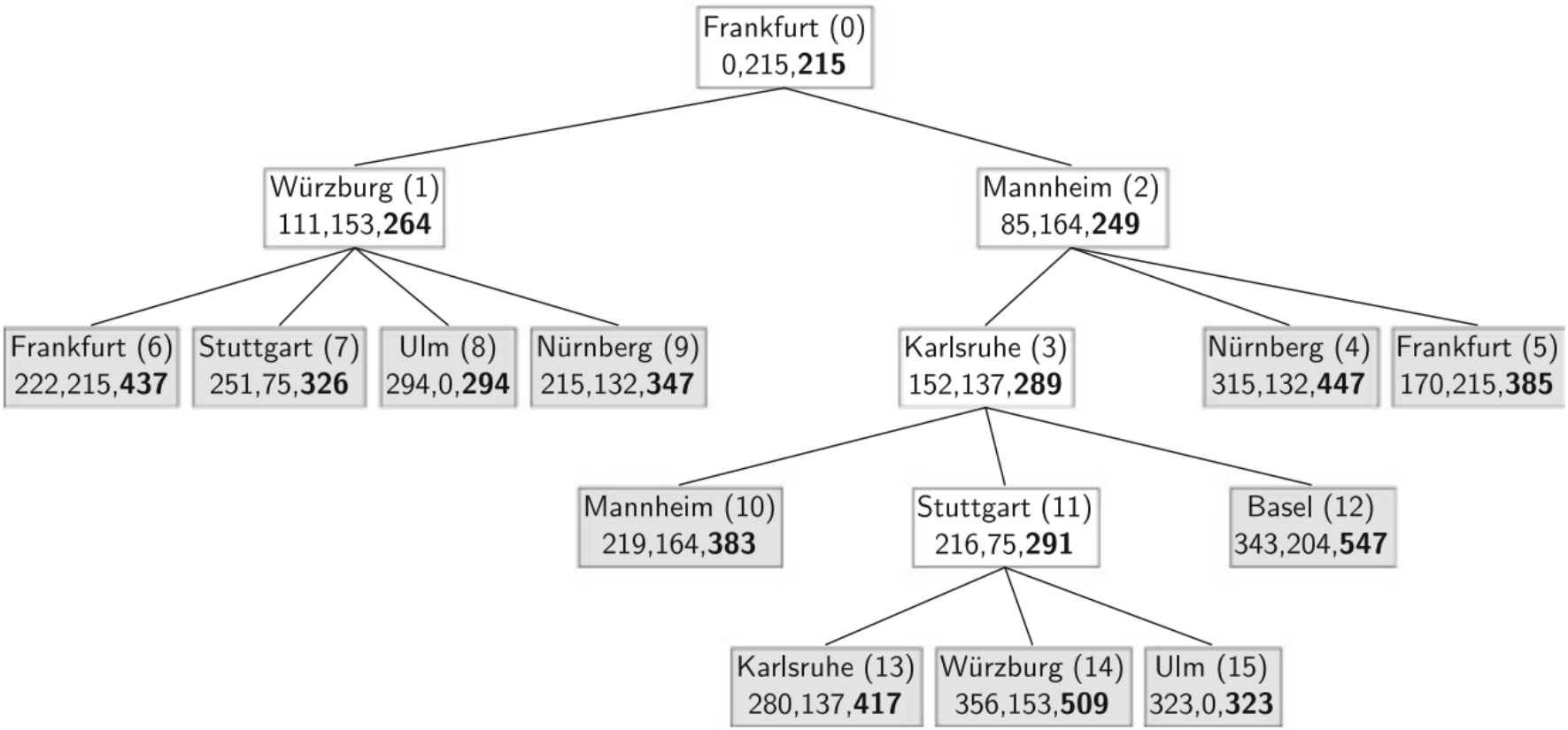
The worst-case time and space complexity for the tree version is $O(b^m)$, where m is the maximum depth of the search space. With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.

A*-Search

The most widely known form of best-first search is called A* search (pronounced "A-star search"). It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal: $f(n) = g(n) + h(n)$.

We take into account the costs that have accurred during the search up to the current node n. First we define the cost function $g(n)$ = Sum of accrued costs from the start to the current node, then add to that the estimated cost to the goal and obtain as the heuristic evaluation function $f(n) = g(n) + h(n)$.

Provided that the heuristic function $h(n)$ satisfies certain conditions, A* search is both complete and optimal.


Frankfurt (0)
0,215,**215**
Würzburg (1)
111,153,**264**
Mannheim (2)
85,164,**249**
Frankfurt (6)
222,215,**437**
Stuttgart (7)
251,75,**326**
Ulm (8)
294,0,**294**
Nürnberg (9)
215,132,**347**
Karlsruhe (3)
152,137,**289**
Nürnberg (4)
315,132,**447**
Frankfurt (5)
170,215,**385**
Mannheim (10)
219,164,**383**
Stuttgart (11)
216,75,**291**
Basel (12)
343,204,**547**
Karlsruhe (13)
280,137,**417**
Würzburg (14)
356,153,**509**
Ulm (15)
323,0,**323**

چه زمانی بهینگی برقرار است؟

The first condition we require for optimality is that $h(n)$ be an admissible heuristic. An admissible heuristic is one that never overestimates the cost to reach the goal.

A heuristic cost estimate function $h(n)$ that never overestimates the actual cost from state n to the goal is called admissible.

The tree-search version of A* is optimal if $h(n)$ is admissible.

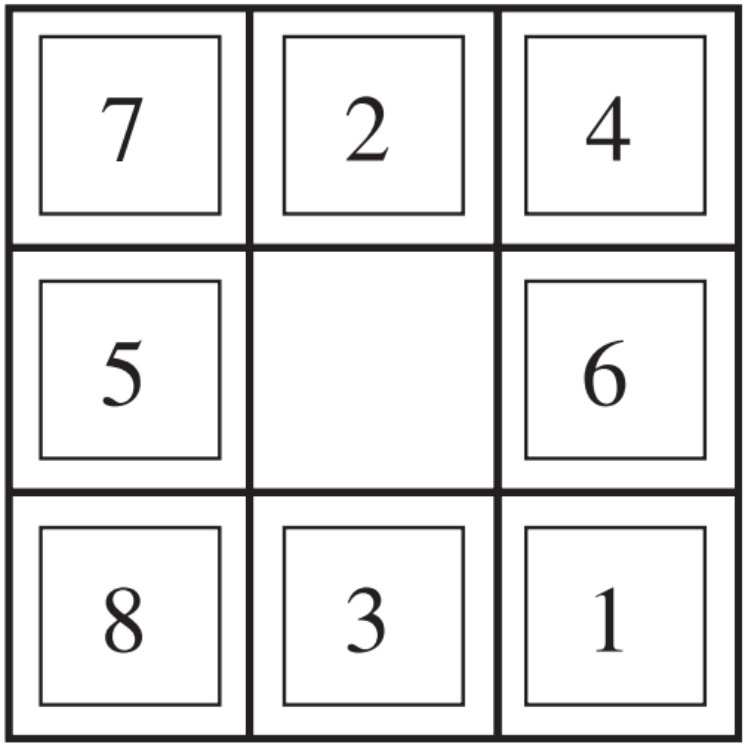
اثبات؟

Example: 8-puzzle

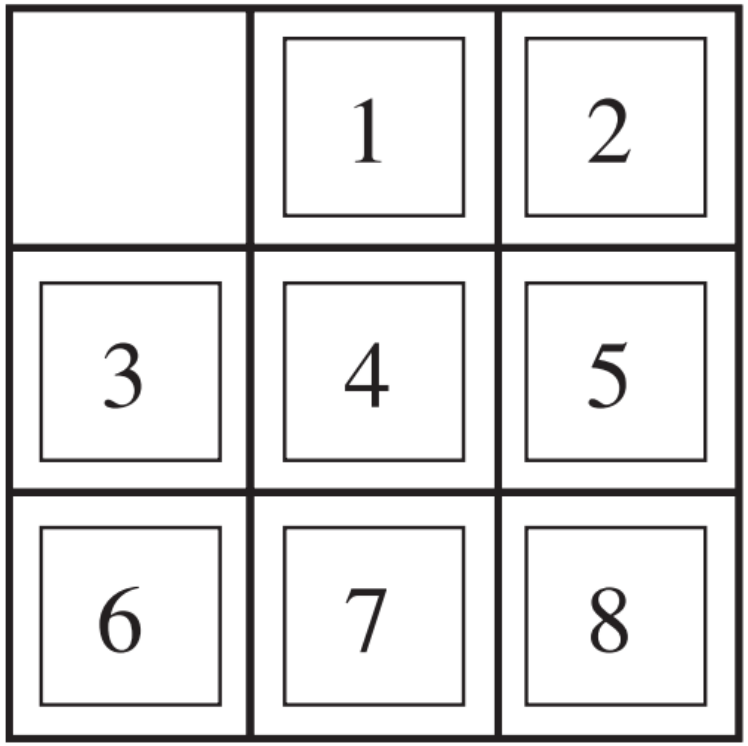
If we want to find the shortest solutions by using A*, we need a heuristic function that never overestimates the number of steps to the goal. Here are two commonly used candidates:

☞ $h_1 =$ **the number of misplaced tiles.**

☞ $h_2 =$ **the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the city block distance or Manhattan distance.**



Start State



Goal State

$$h_1 = 8 \qquad h_2 = 18$$