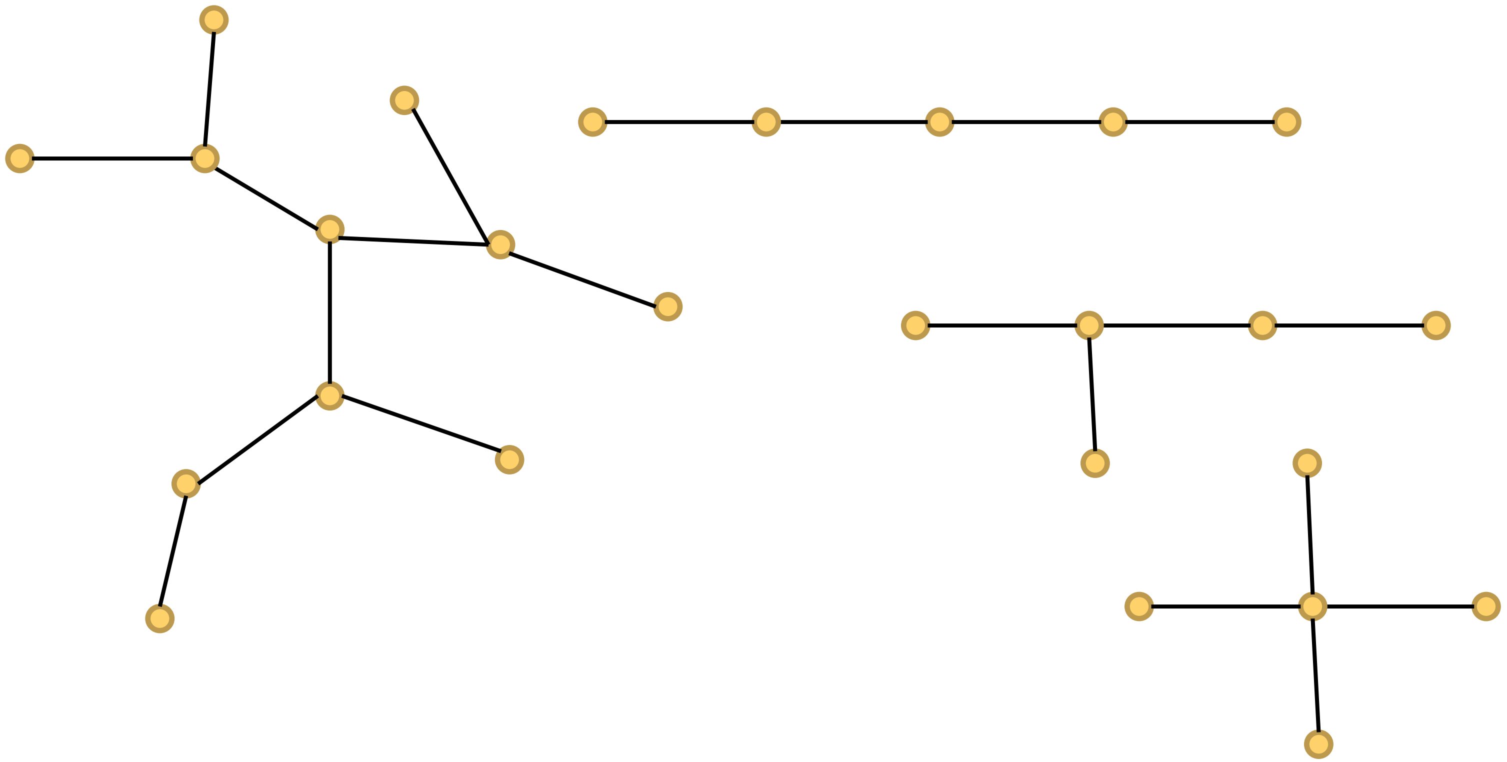
Computer Science Fundamentals

Graph Theory – Trees

Technische Hochschule Rosenheim
Winter 2021/22
Prof. Dr. Jochen Schmidt

Forest

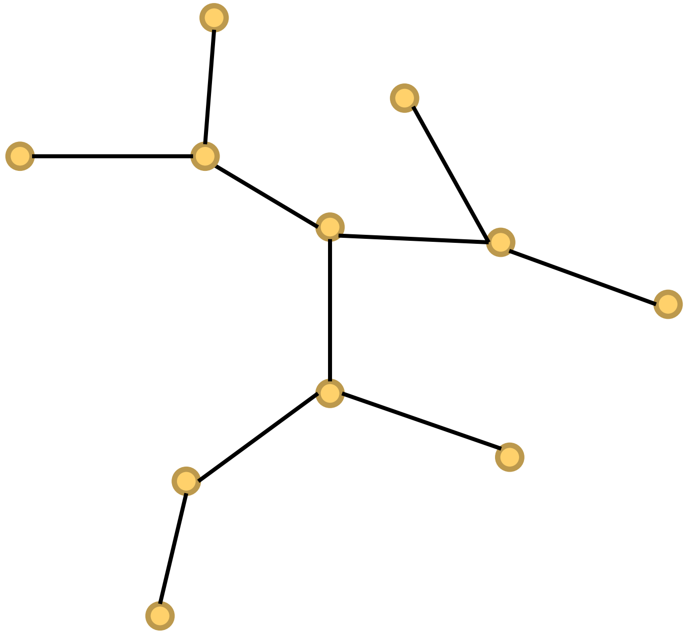
A (simple, undirected) graph G is called forest (*Wald*)
if and only if G does not contain closed trails



Tree

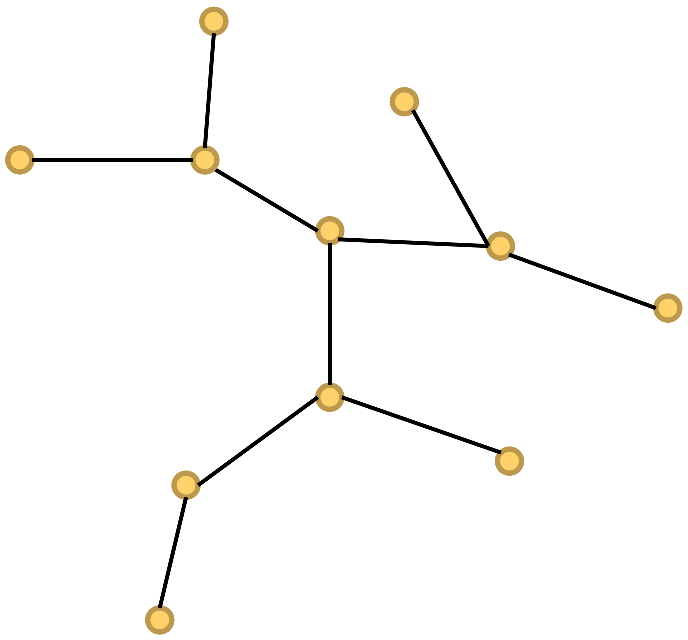
Technische Hochschule Rosenheim

G is called a tree if and only if G is a connected forest



Properties of Trees

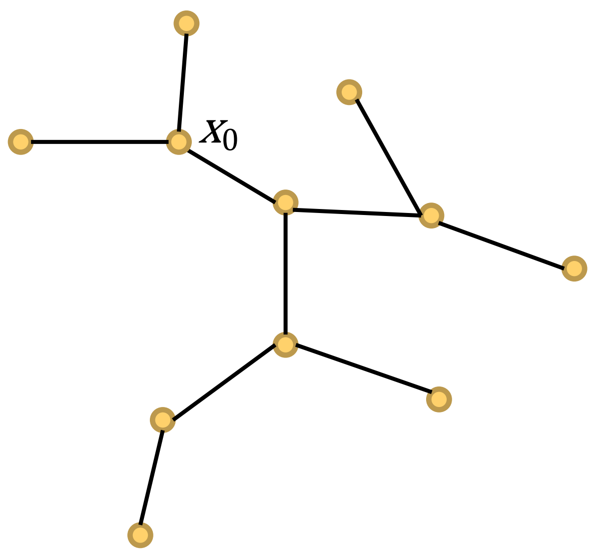
- each pair of vertices is connected by exactly one path
- if you remove an edge, the graph is no longer connected
- a tree with n vertices has exactly $n - 1$ edges

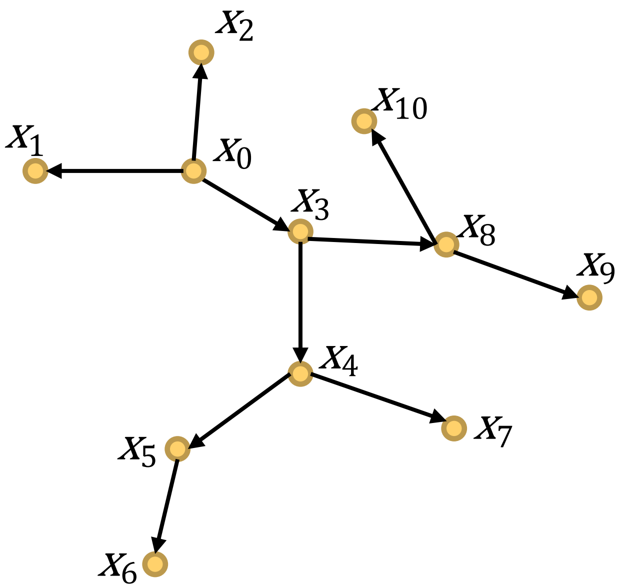


Rooted Trees

We obtain a rooted tree (*Wurzelbaum*) if

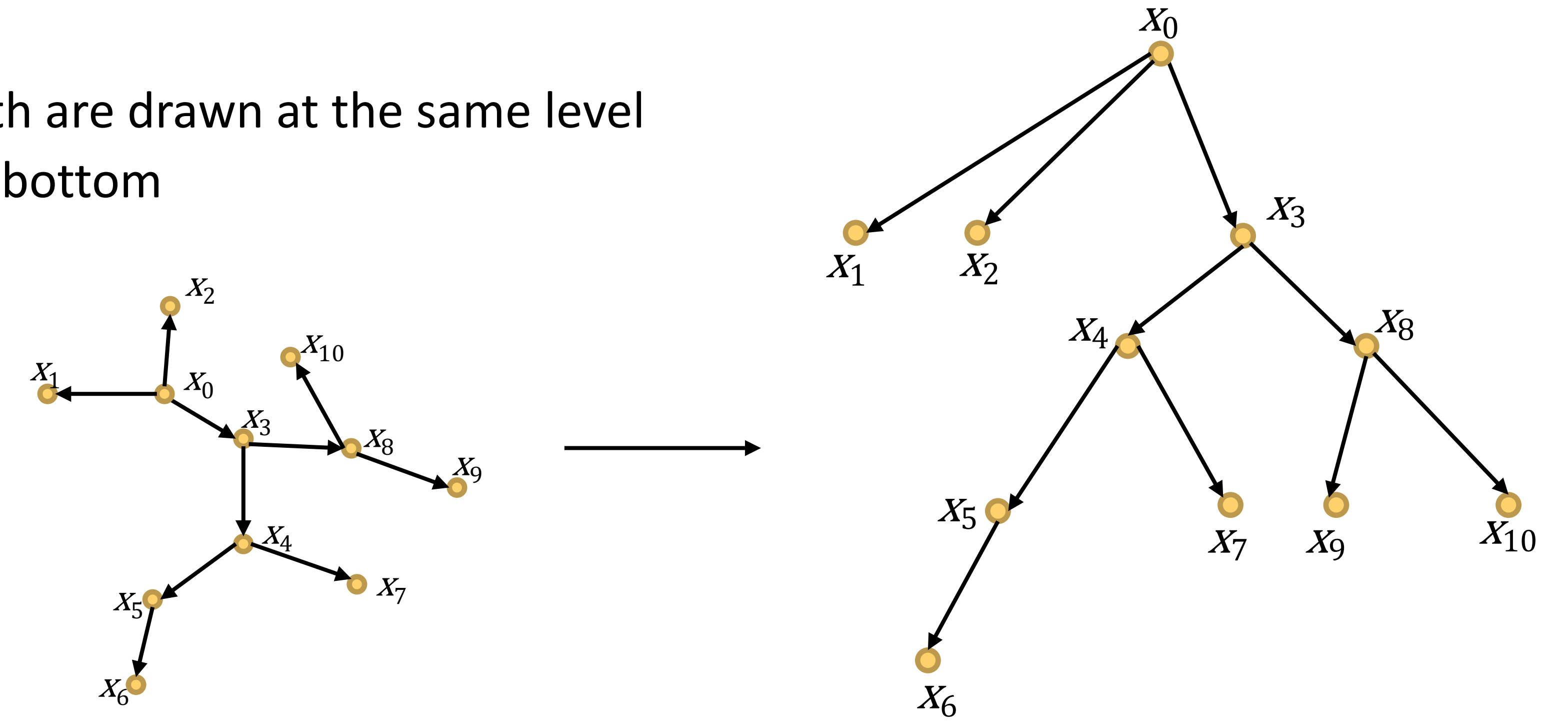
- we choose one vertex x_0 of a tree as root (*Wurzel*)
- replace all undirected edges $[x, y]$ by directed (x, y) edges where $d(x_0, x) < d(x_0, y)$





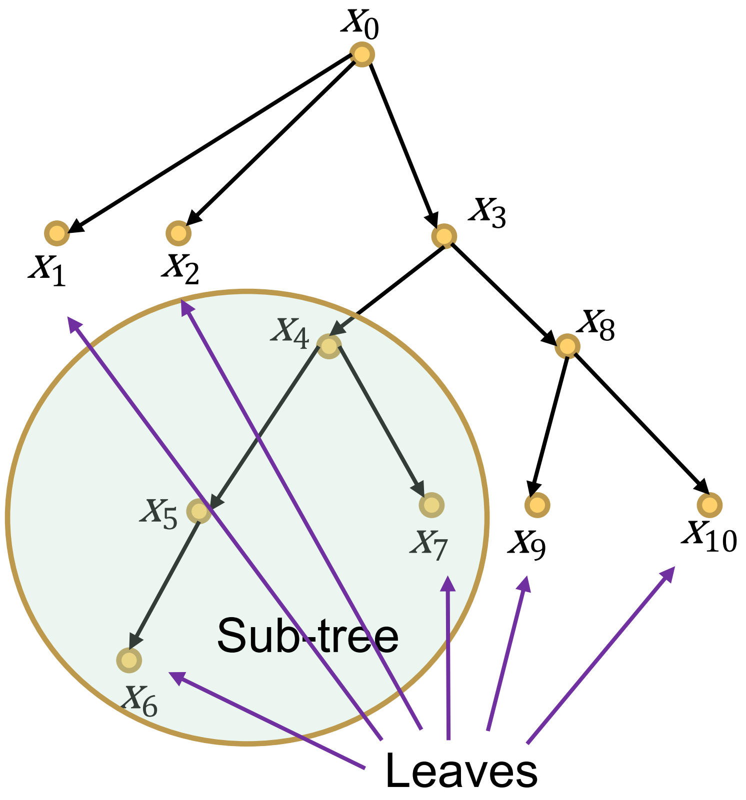
Rooted Trees

- $d(x_0, x)$ is called the depth (*Niveau*) of vertex x
- typically
  - the root is drawn at the top
  - all vertices having the same depth are drawn at the same level
  - thus, the tree grows from top to bottom
- It holds:
  $d^-(x_0) = 0$
  $d^-(x_i) = 1 \;\forall\; i \neq 0$



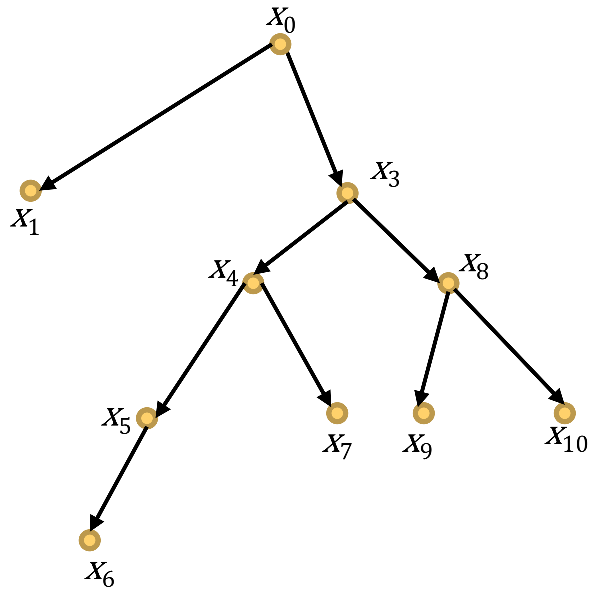
Rooted Trees

- Sub-tree (*Teilbaum, Unterbaum*): Set of paths accessible from a vertex
- Leaves (*Blätter*): Vertices with $d^+(x_i) = 0$, i.e. the end nodes



Special Rooted Trees

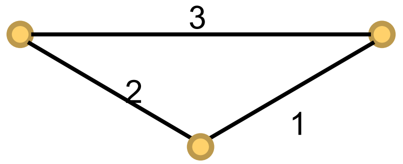
- Binary tree (*Binärbaum*): $d^{+}(x_i) \leq 2$ holds for all vertices

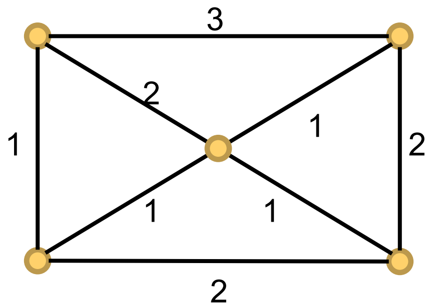


- List: $d^{+}(x_i) \leq 1$ holds for all vertices

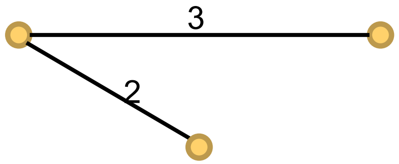
Spanning Forest

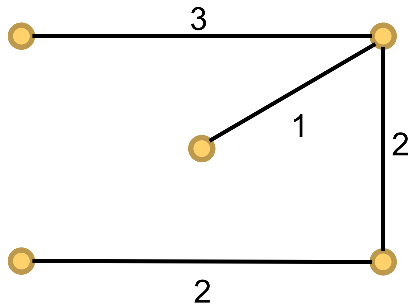
- Spanning forest W (*Gerüst, aufspannender Wald*) of a graph G
  - W is a forest
  - the vertex sets of W and G match



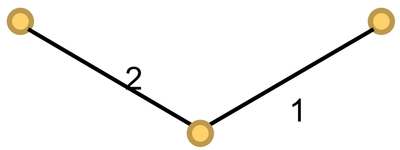


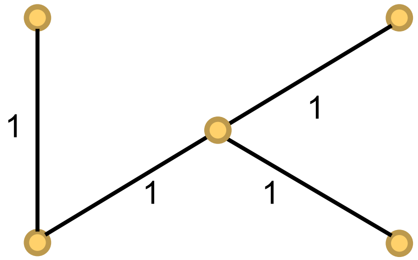
- W is created by deleting edges
  - Connected components remain intact
  - there are no closed trails





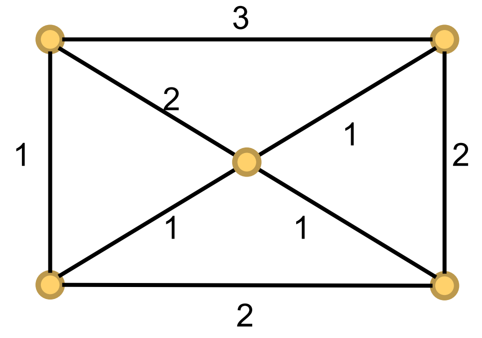
- Minimal Spanning forest (*Minimalgerüst*)
  - in weighted graphs
  - the spanning forest where the sum of edge weights is minimal

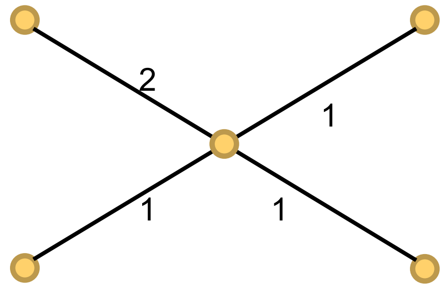




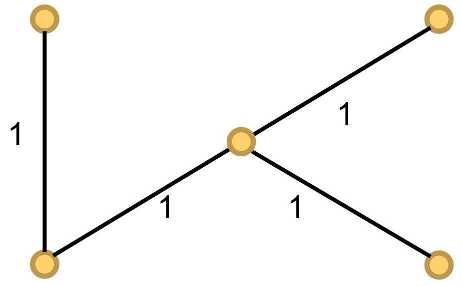
Spanning Tree

- Spanning tree (*Spannbaum*)
  - spanning forest of a connected graph



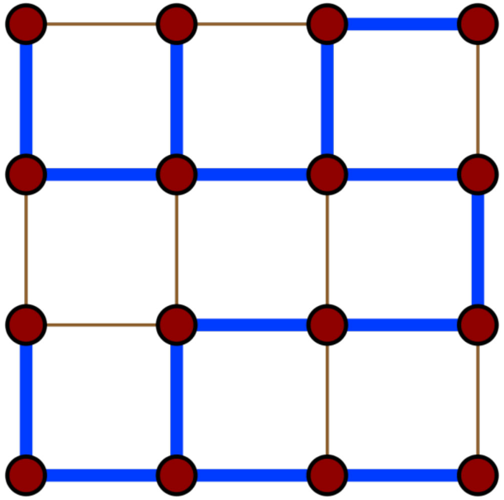


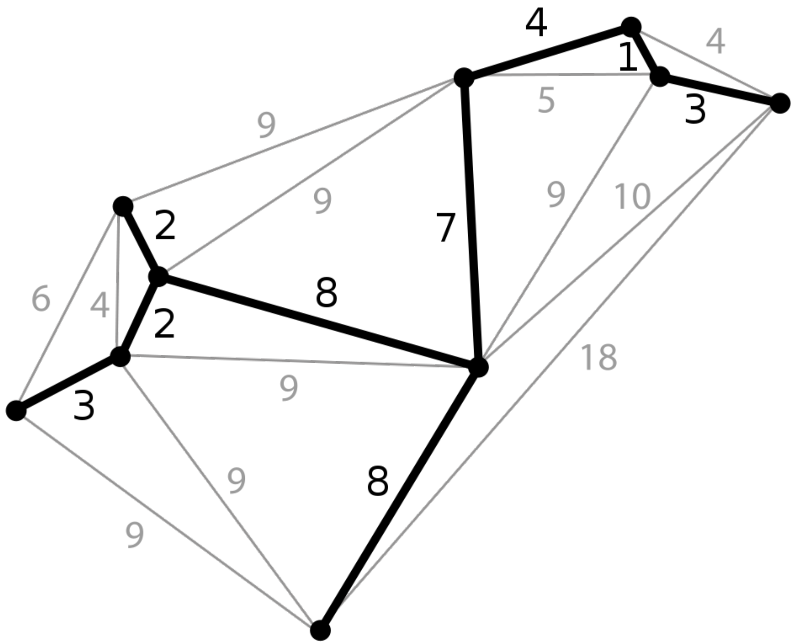
- Minimal spanning tree
  - minimal spanning forest of a connected graph



- Application example
  - Design of cost-effective networks (telephone, electricity, computer)
- Algorithms
  - Spanning tree: Depth-/Breadth-First Search
  - Minimal spanning tree: Kruskal, Prim ⟶ course „Agorithms & Data Structures“

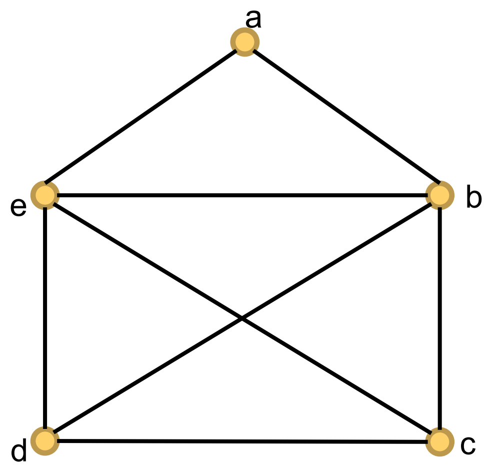
Spanning Tree – Examples



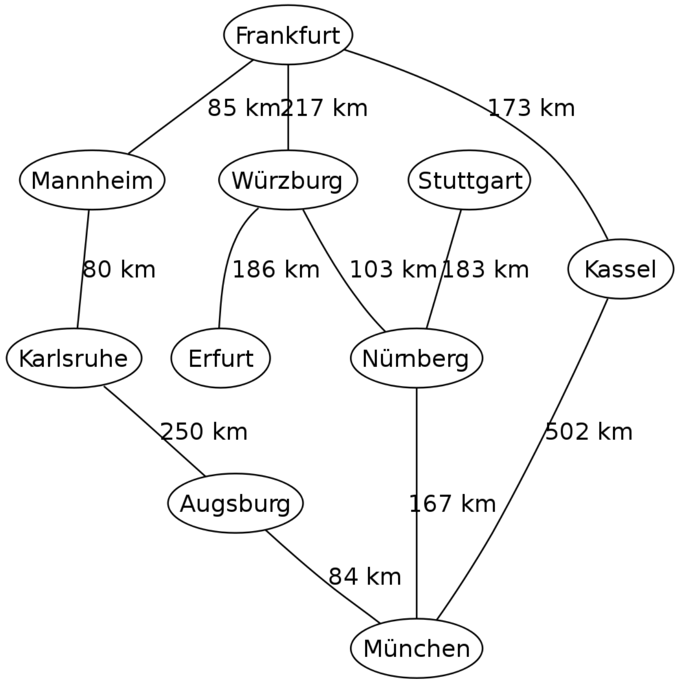


Exercise

Draw 3 non-isomorphic spanning trees of the following graph:



Graph Search Using Trees – Motivation



Typical questions:

- how do I get from Frankfurt to Munich?
- what is the shortest route from Frankfurt to Munich?

Search strategy: Determined by the order of node expansion

Idea: **Generate a tree to search** the route

1. Initialize tree with start node
2. Repeat until target node expanded or search fails:
   - **if** no candidates for expansion **then return** FAILED
   - Select leaf to be expanded based on your search strategy
     - mark this node as "closed" – it will not be added to the tree during further processing any more
   - **if** selected leaf is the target node **then return** SOLUTION
   - **else** expand leaf by adding neighboring nodes to tree

Evaluation Criteria for Graph Search Algorithms

- Criteria:
  - Completeness: *Is a solution always found, if it exists?*
  - Optimality: *Is the solution with the lowest cost always found?*
  - Time complexity: Computation time required: *Number of generated/expanded nodes*
  - Space complexity: Memory required: *Number of nodes stored during search*

Time & space complexity will be covered in detail in Theoretical Computer Science, 2nd semester

- Time and space complexity are measured based on the difficulty of the problem, defined by:
  - *b – Maximum degree of branching of the search tree*
  - *d – Depth of solution having lowest costs*
  - *m – Maximum depth of the state space (may be infinite)*

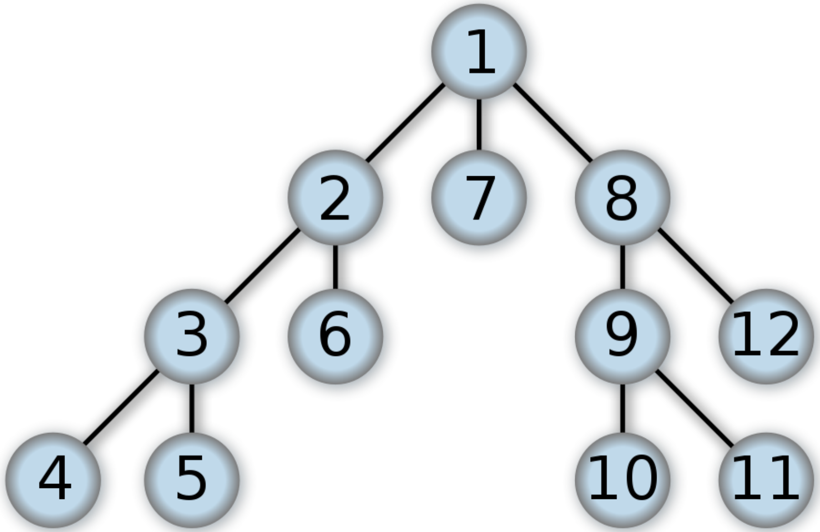
Uninformed Search Strategies

- Uninformed search = blind search
  - use only information that is available in the problem definition
  - if a strategy can determine whether a non-target node is better than another ⟶ informed search

- Search strategies:
  - Depth-first Search (*Tiefensuche*)
  - Breadth-first Search (*Breitensuche*)
  - Uniform-cost Search (*uniforme Kosten Suche*)

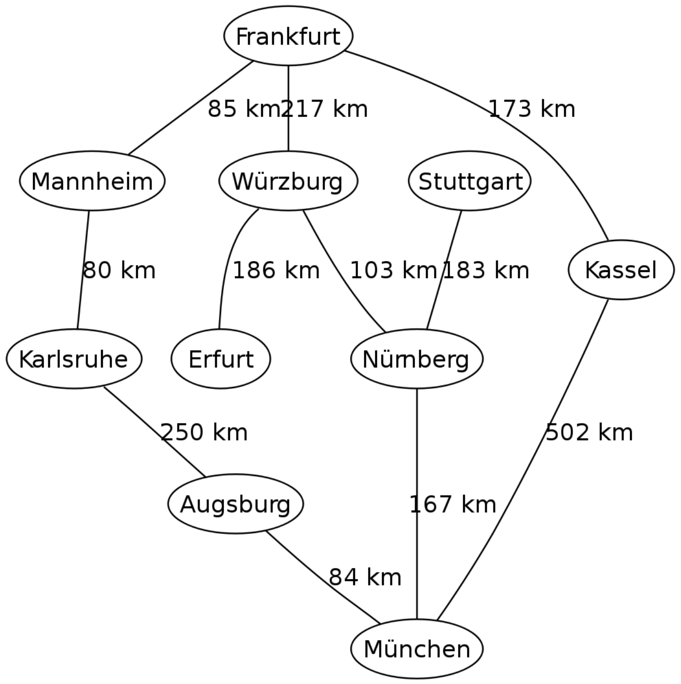
Depth-first Search (*Tiefensuche*)

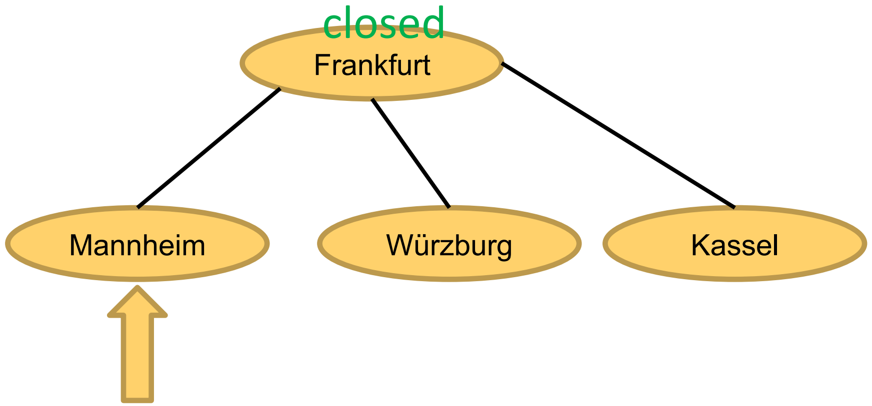
- Strategy: expand the deepest leaf
- Data structure for leaves: Stack (LIFO)
- This is the algorithm you can use for walking through a maze from entry to exit
  - by always keeping one hand to a wall
  - you do not need a map



Depth-first Search – Example

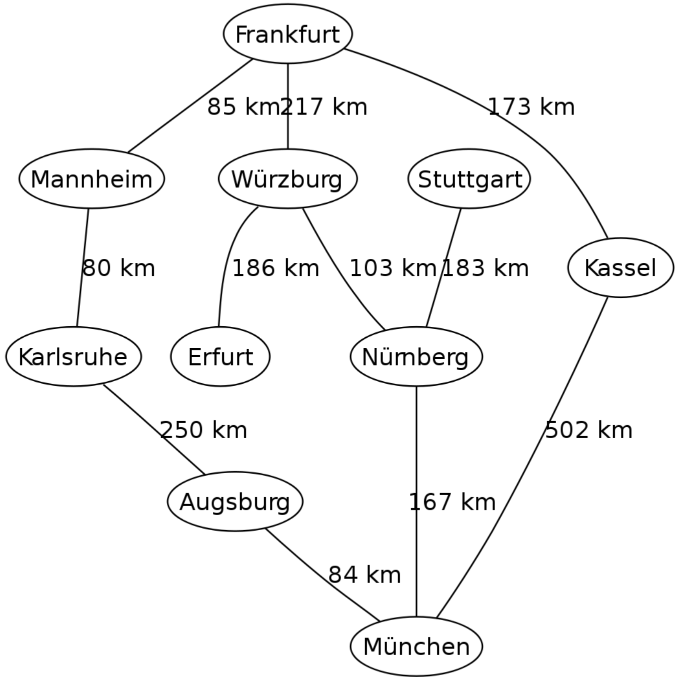
How do I get from Frankfurt to München?

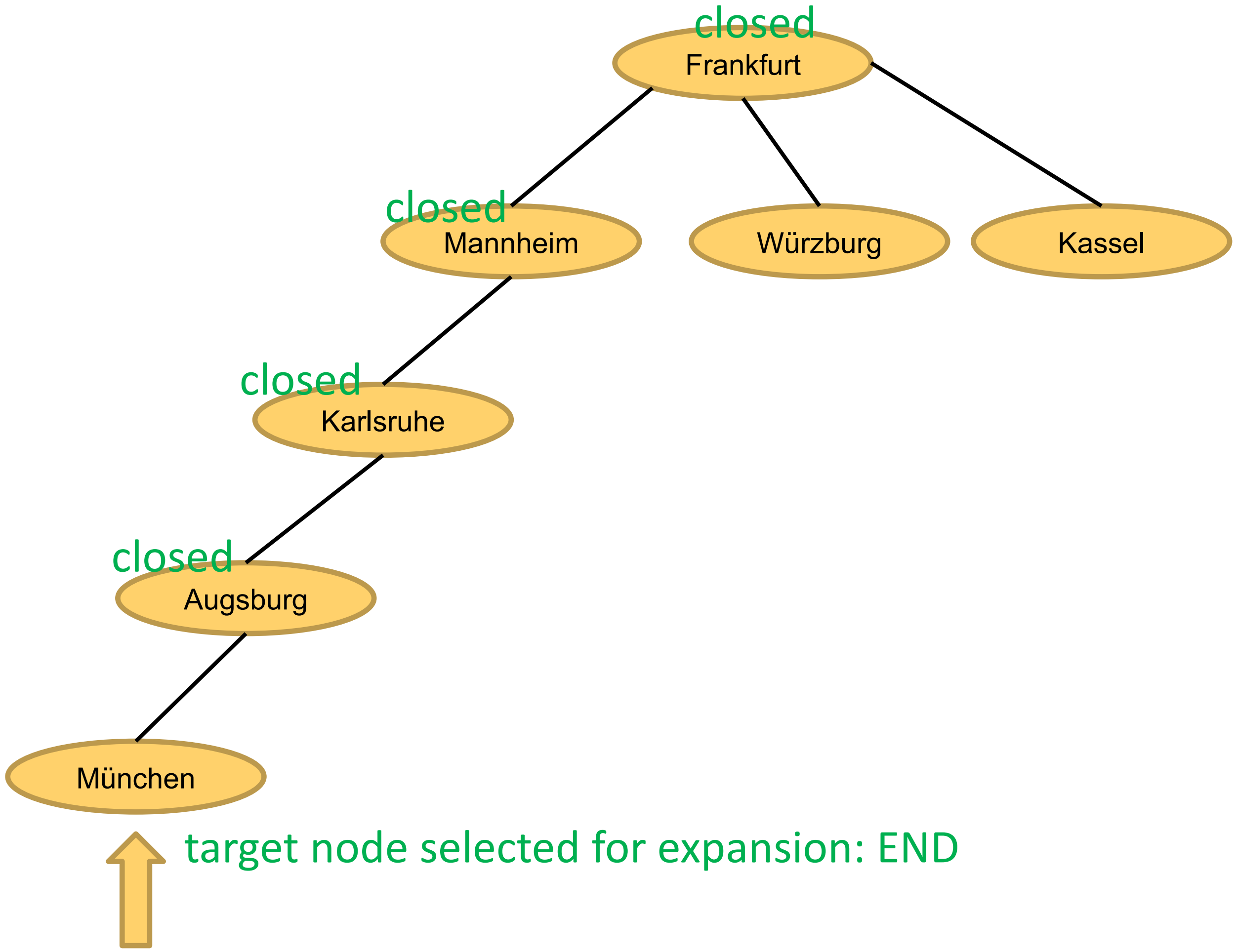




Depth-first Search – Example

How do I get from Frankfurt to München?





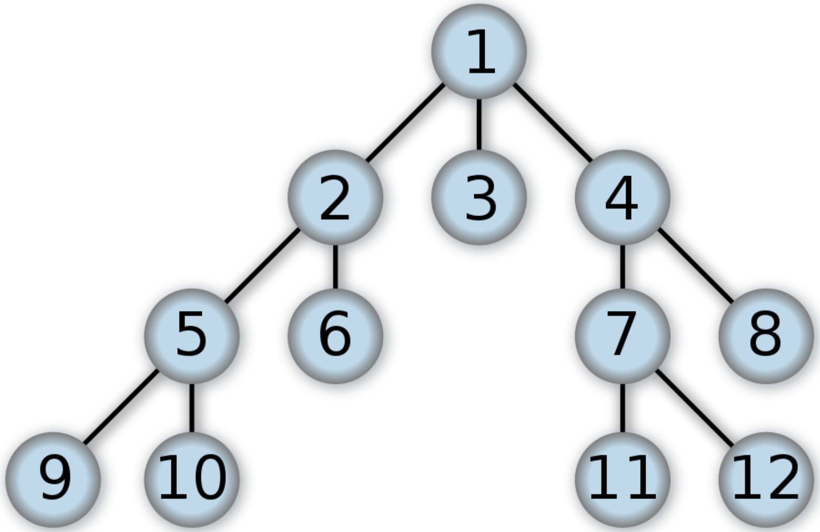
is this the shortest route?

Depth-first Search – Evaluation

- Completeness:
  - in general: NO
  - if the search space is finite and there are no closed trails: YES
- Optimality:
  - NO
- Time complexity: $O(b^m)$
  - very bad if m (maximum depth of the search space) is much larger than d (depth of the optimal solution)
  - but: faster than breadth-first search, if there are several solutions
- Space complexity: $O(bm + 1)$

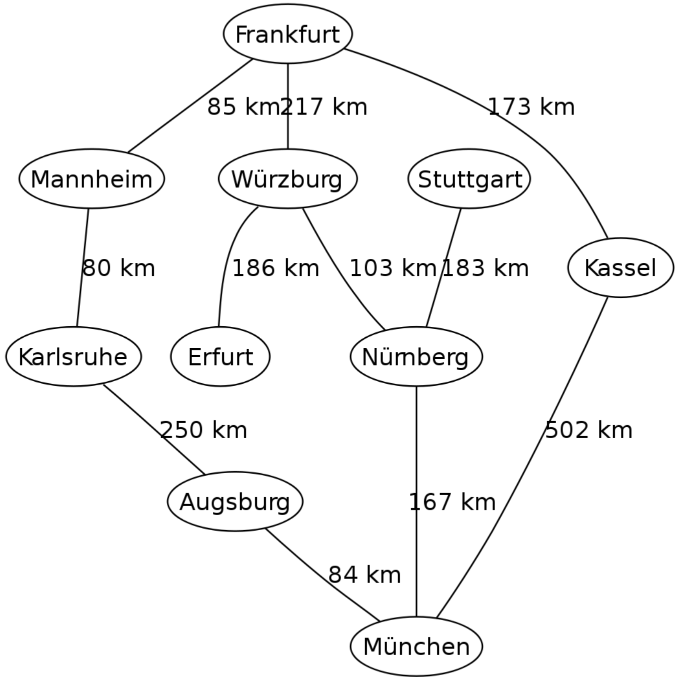
Breadth-first Search (*Breitensuche*)

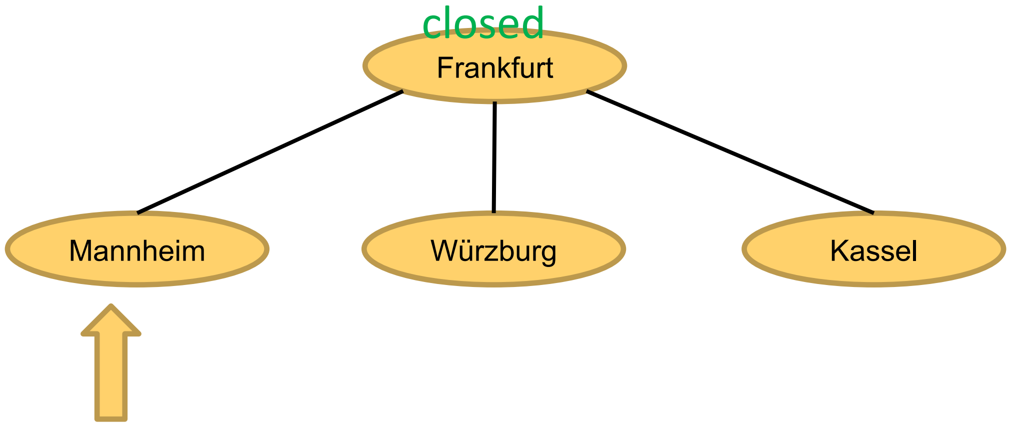
- Strategy: expand the shallowest leaf
- Data structure for leaves: Queue (FIFO)
- You can use this algorithm for walking through a maze from entry to exit
  - but you do need a map



Breadth-first Search – Example

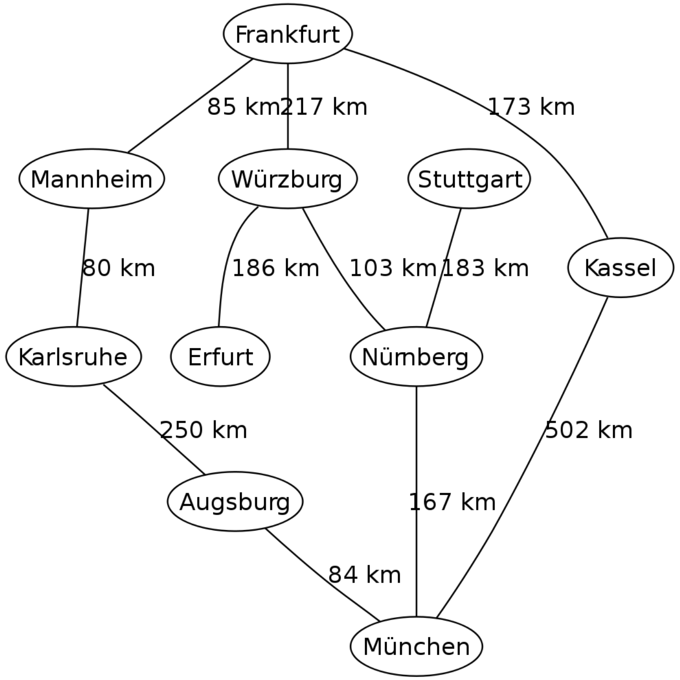
How do I get from Frankfurt to München?

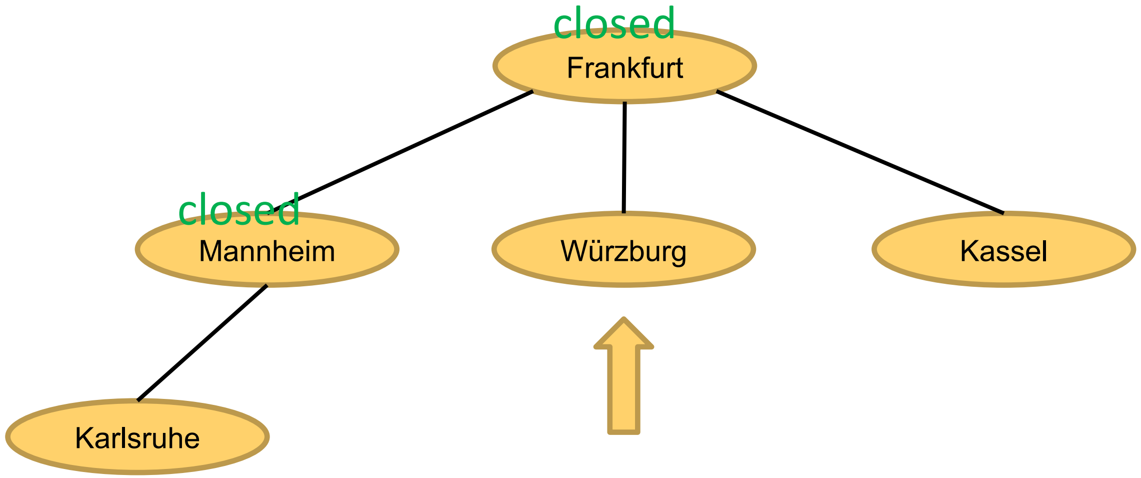




Breadth-first Search – Example

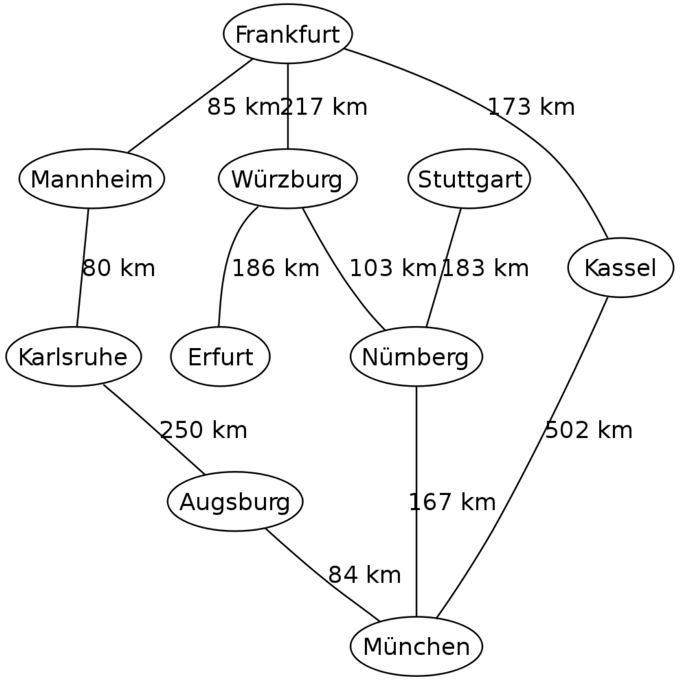
How do I get from Frankfurt to München?

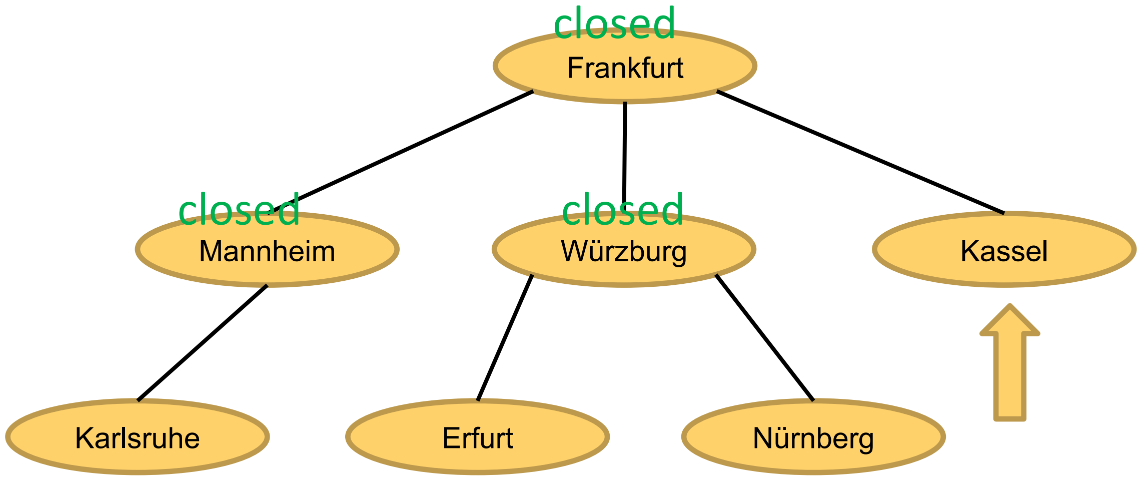




Breadth-first Search – Example

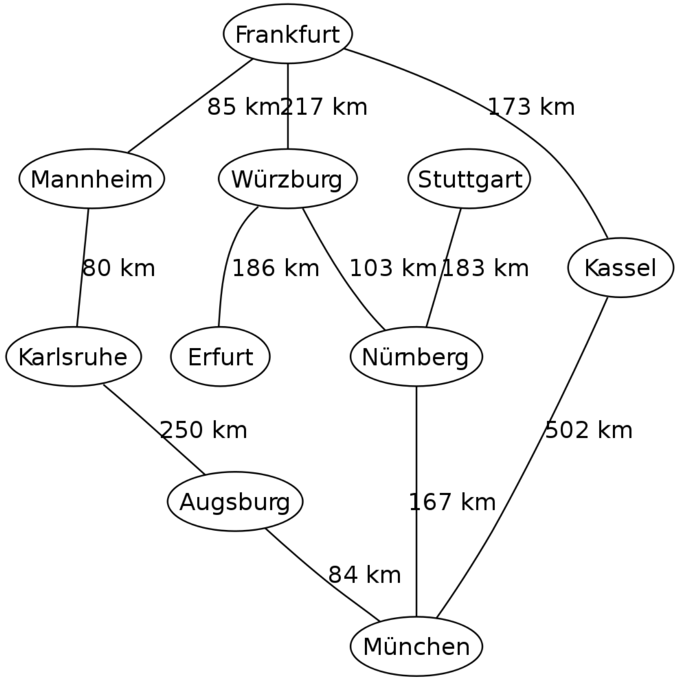
How do I get from Frankfurt to München?

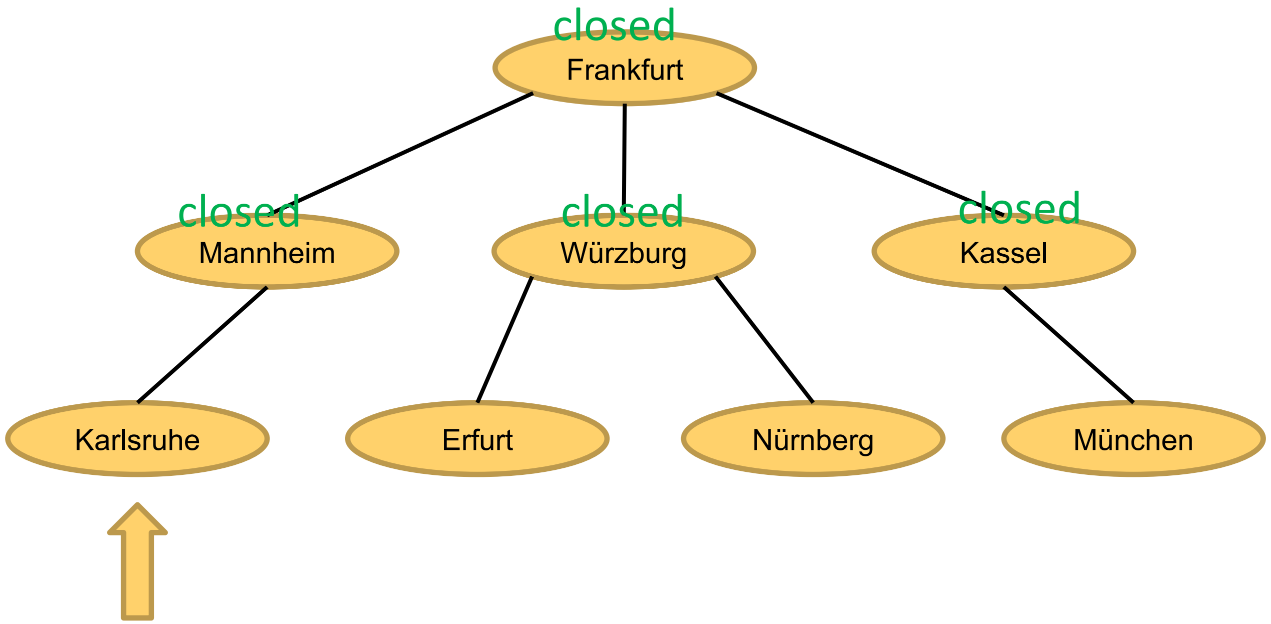




Breadth-first Search – Example

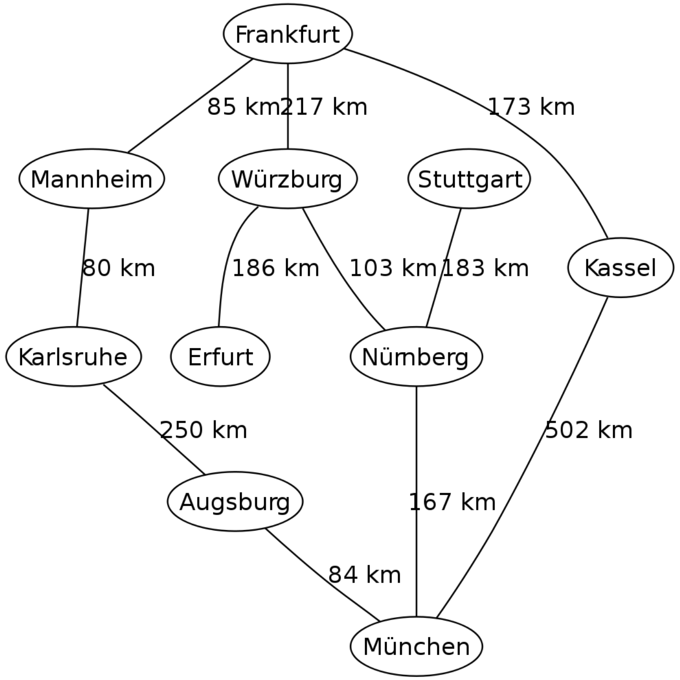
How do I get from Frankfurt to München?

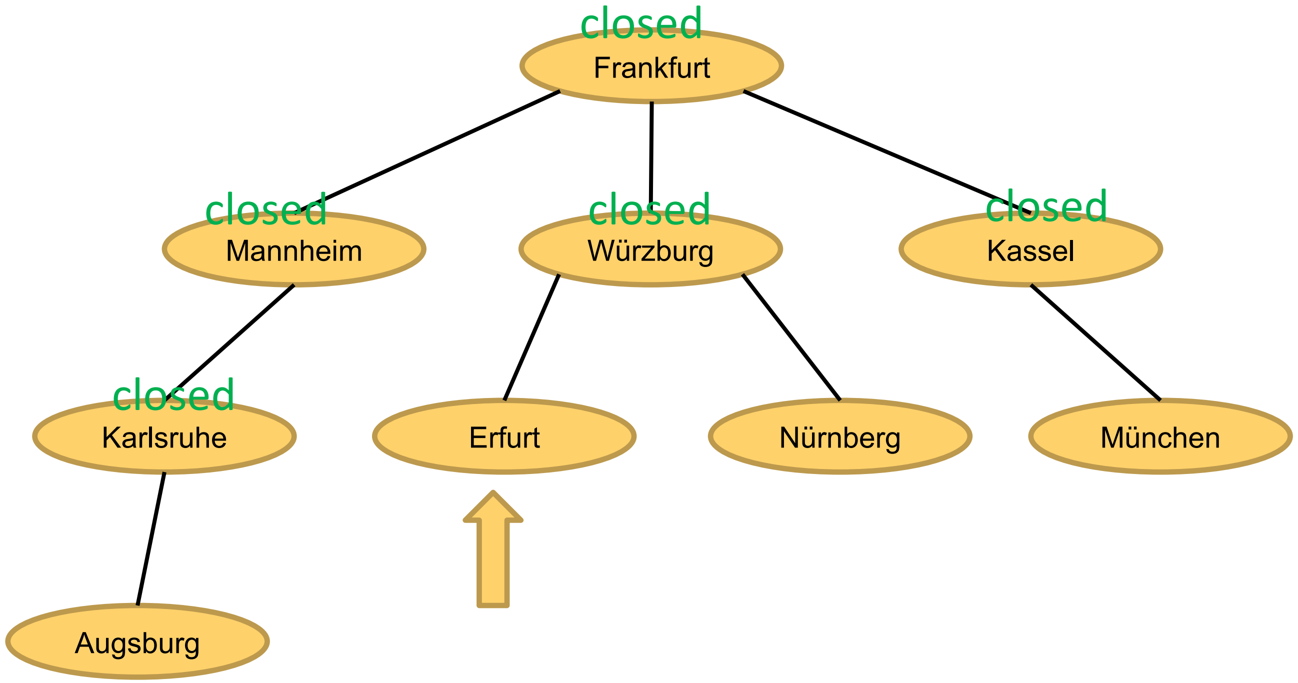




Breadth-first Search – Example

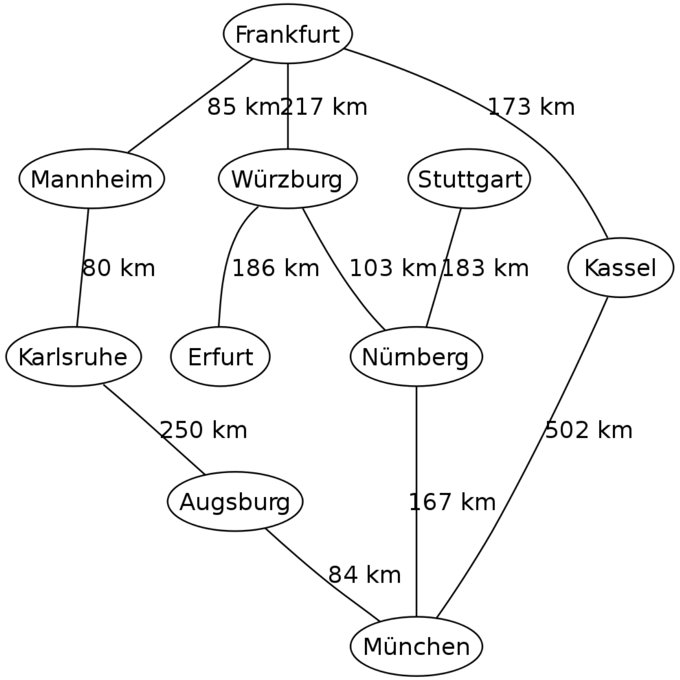
How do I get from Frankfurt to München?

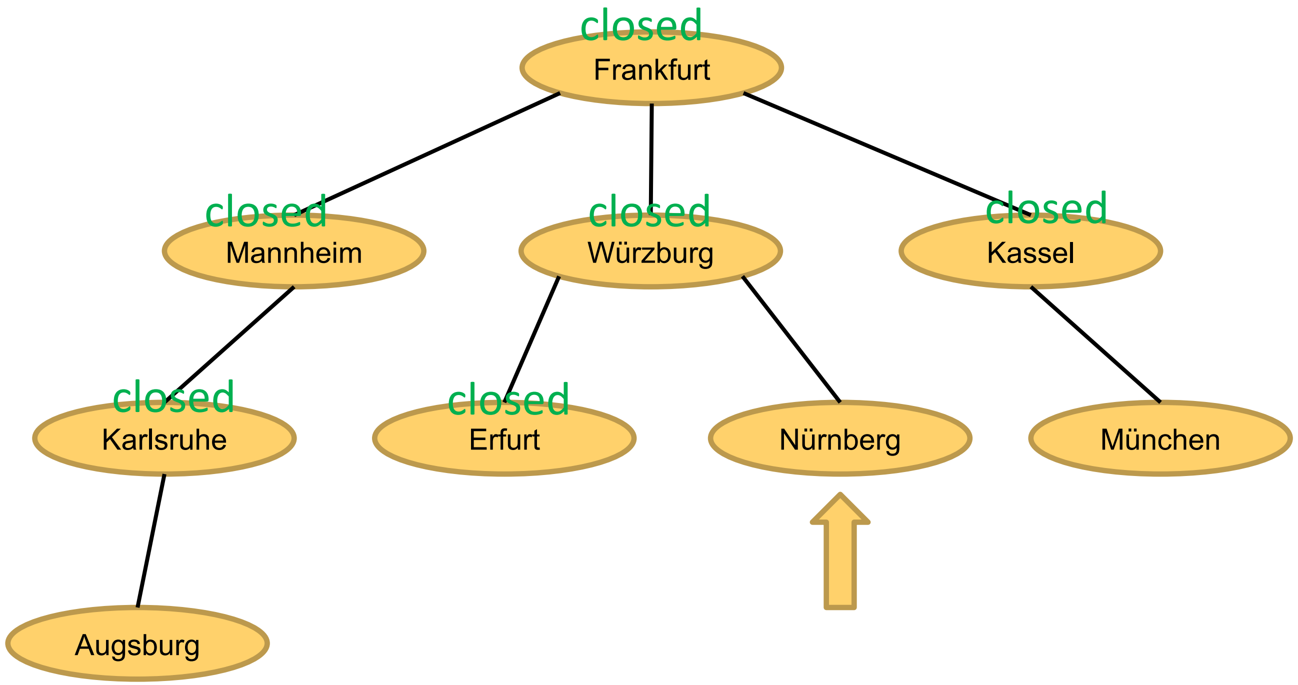




Breadth-first Search – Example

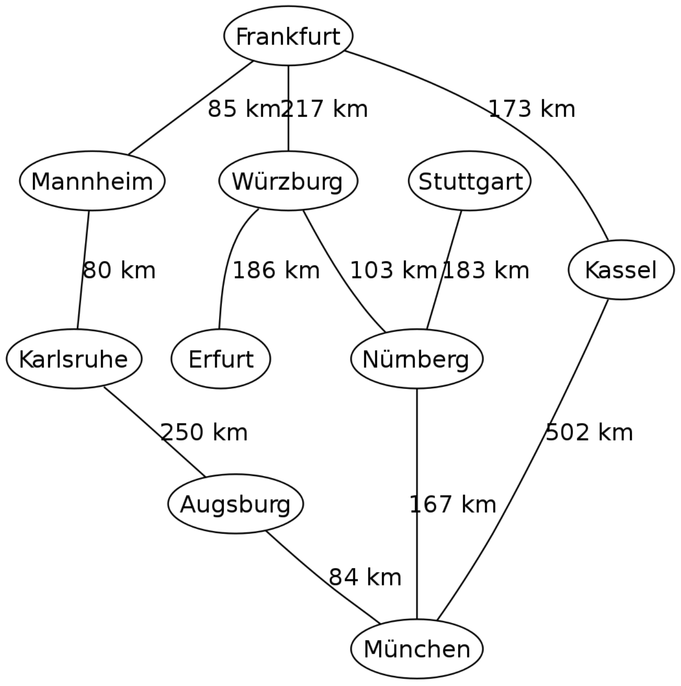
How do I get from Frankfurt to München?

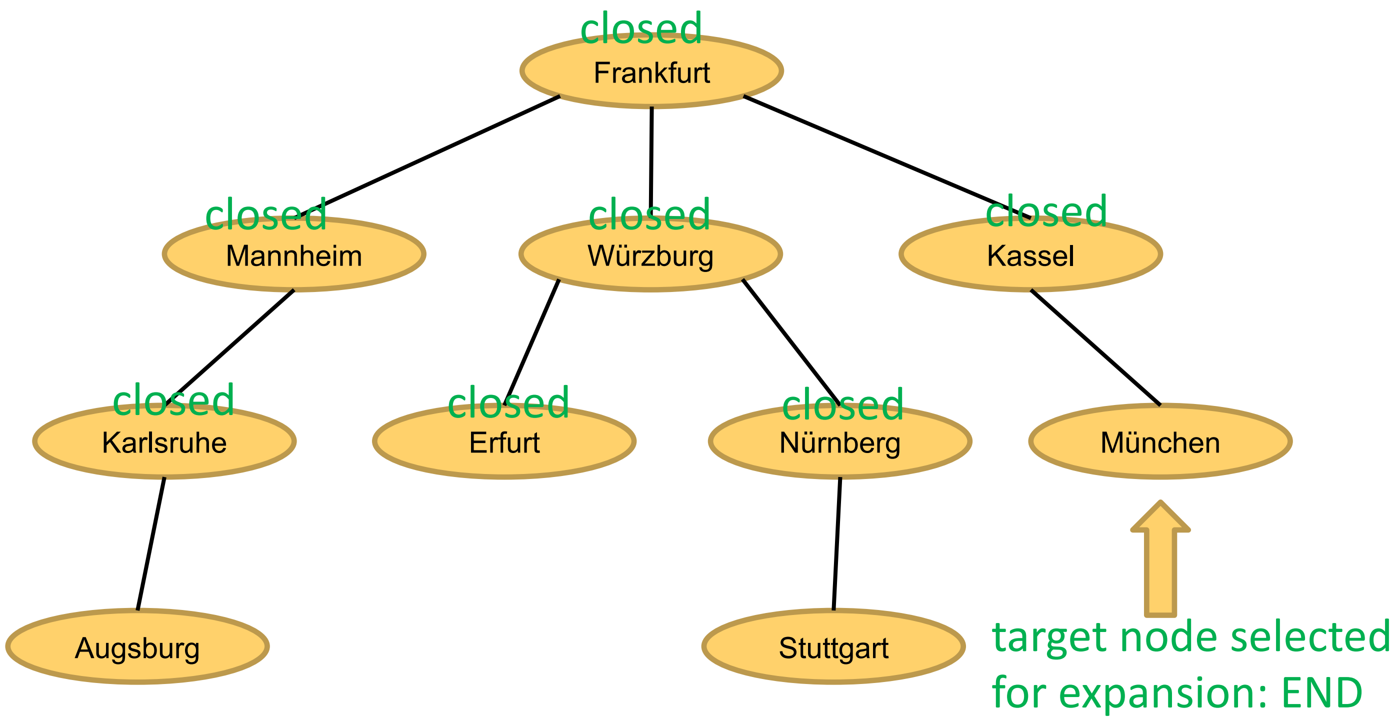




Breadth-first Search – Example

How do I get from Frankfurt to München?





is this the shortest route?

Breadth-first Search – Evaluation

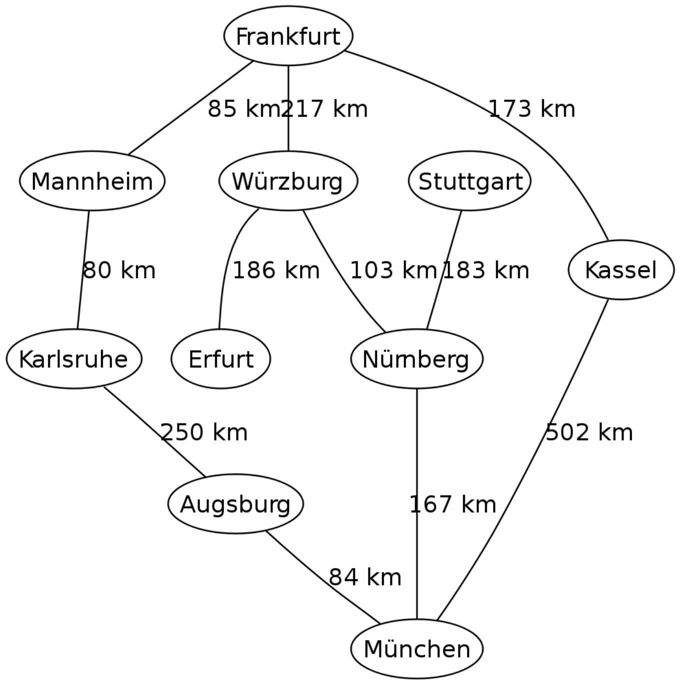
- Completeness: YES
  - if the shallowest target node is at finite depth d
  - and b (maximum number of successors of a node) is finite
- Optimality:
  - YES, but only regarding minimum number of edges (i.e., if all edges have the same weight)
- Time complexity: $O(b^{d+1})$
  - Let each node have b successors
  - Root has b successors, each node of the next level has again b successors (total: b^2), …
  - Suppose the solution is at depth d
  - Worst case: all nodes except the last one at depth d have to be expanded
  - Total number of nodes generated: $b + b^2 + b^3 + \cdots + b^d + \left(b^{d+1} - b\right) = O(b^{d+1})$
- Space complexity: same as time complexity, all nodes have to be kept in memory

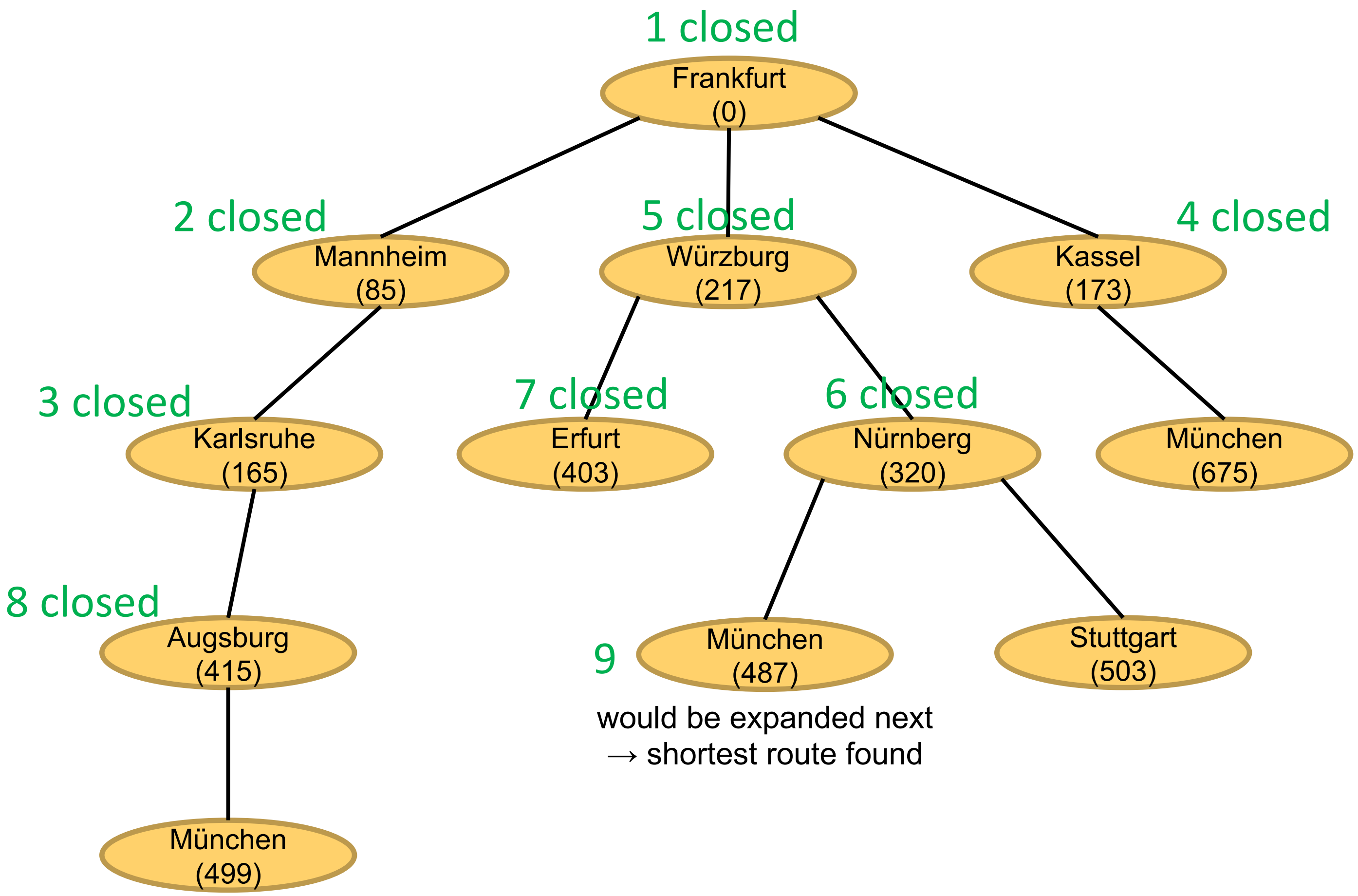
Uniform-cost Search

- Extension of breadth-first search for weighted graphs
- Strategy
  - each node gets assigned the total cost from the root
  - expand the node that has the lowest total cost
- Data structure for leaves: Priority Queue (*Prioritätswarteliste*)
- This is basically Dijkstra's algorithm (which determines all shortest paths)

Uniform-cost Search – Example

How do I get from Frankfurt to München?





Uniform-cost Search – Evaluation

- Completeness: YES (same as breadth-first)
- Optimality:
  - YES, if weights are non-negative
- Time complexity: $O(b^{1+c/e})$
  - c: Cost of the optimal solution
  - e: minimum possible (positive) edge weight
- Space complexity: same as time complexity, all nodes have to be kept in memory

Uniform-cost/Breadth-first Search – Evaluation

- Memory requirements are often a bigger problem than runtime
- Search problems with exponential complexity can be solved by uninformed search algorithms only for very small problems
- Example: Breadth-first Search
  - Assumption: Branching factor b = 10; 100 000 nodes/sec; 1000 byte/nodes

Solution at depth	#expanded nodes	Time	Memory
2	1100	0,011 sec	1 MB
4	111100	1,1 sec	106 MB
6	10^7	1,9 min	10 GB
8	10^9	3,1 h	1 TB
10	10^{11}	13 Tage	101 TB
12	10^{13}	3,5 Jahre	10 Petabyte
14	10^{15}	352 Jahre	1 Exabyte

Informed Search

- Idea: Best-first search
  - node to be expanded is selected using an evaluation function $f(n)$
  - avoid expansion of paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ real costs from root to current node n
  - $h(n)$ estimated costs from node n to target node
  - $f(n)$ estimated total costs from start to target via node n

Special cases:

- uniform-cost search ($h(n) = 0$): $f(n) = g(n)$
- greedy search ($g(n) = 0$): $f(n) = h(n)$

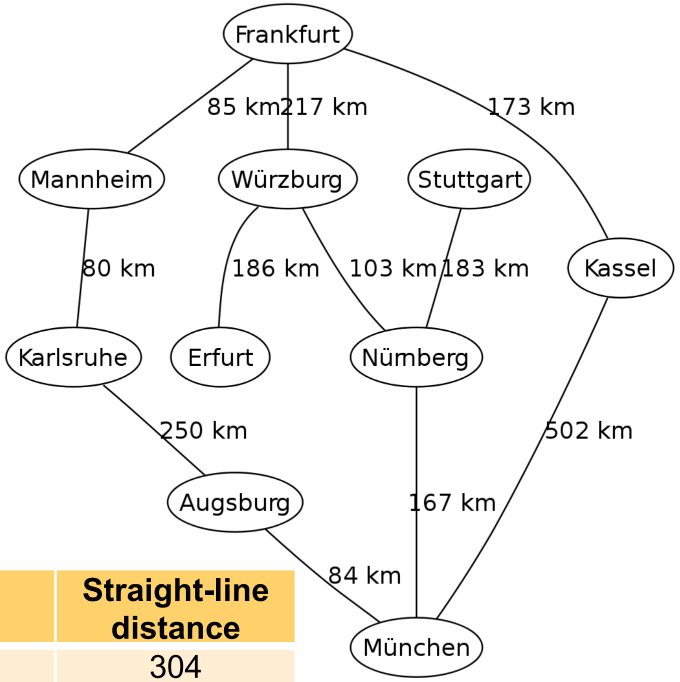
- Select the node having lowest total costs for expansion
  - data structure: priority queue
- $h(n)$ is called the heuristic function
  - $h(n)$ must provide additional information,
  - i.e., it must not be calculable from the graph itself

A* Search

- A* uses an optimistic heuristic
  - the costs to the target node are never overestimated
  - $h(n) \leq h^*(n)$, where $h^*(n)$ are the actual cost from n to the target node
- usually: use monotonous heuristics
  - even more restrictive than “optimistic”
  - additionally: $h(n) \leq t(n, n') + h(n')$
    - n and n' are adjacent nodes
    - $t(n, n')$ are the actual costs from n to n'
    - the estimated costs to the target must be less than the costs via any neighboring node (triangle inequality)
  - Euclidean distance is monotonous

A* Search – Example

Shortest path from Frankfurt to München?



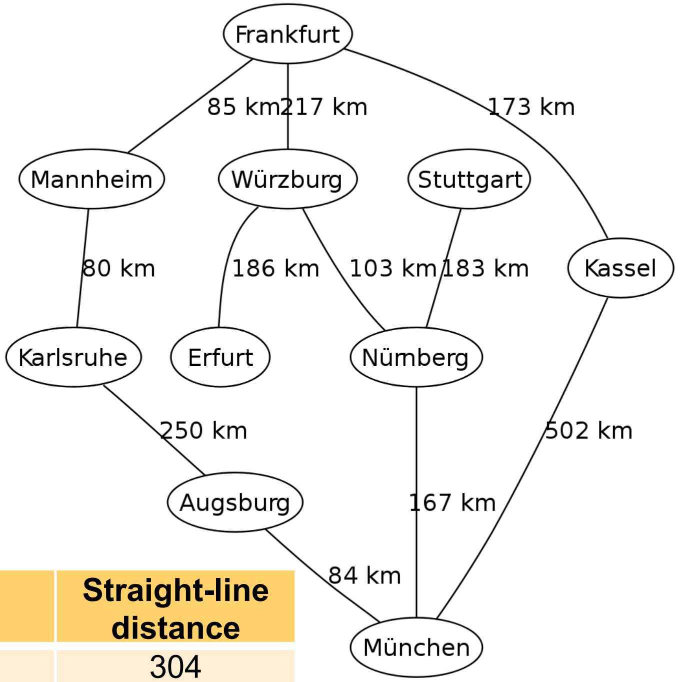
City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0

Frankfurt
(304)

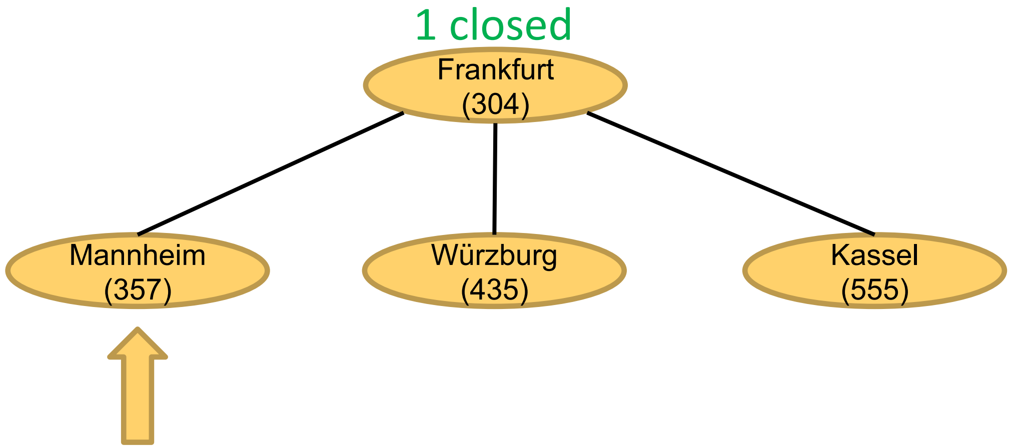
$f(F) = g(F, F) + h(F) = 0 + 304 = 304$

A* Search – Example

Shortest path from Frankfurt to München?



City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0



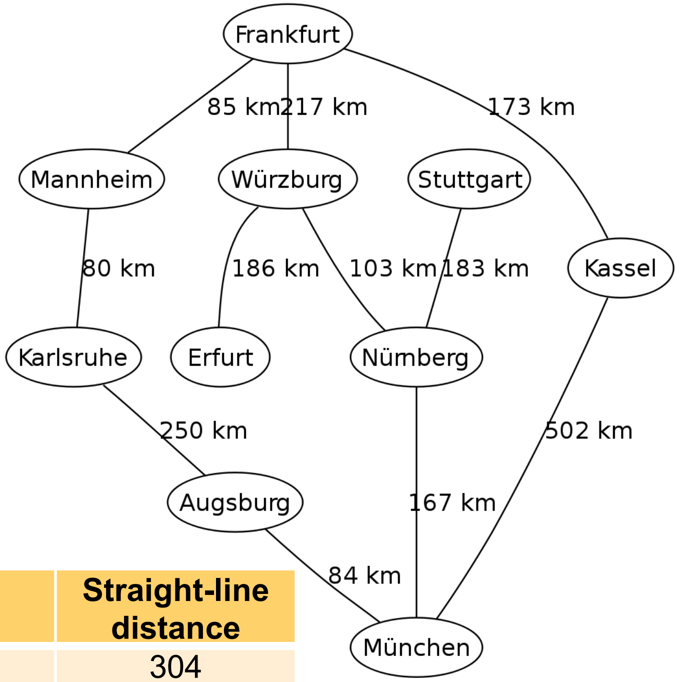
f(MA) = g(F, MA) + h(MA) = 85 + 272 = 357

f(WÜ) = g(F, WÜ) + h(WÜ) = 217 + 218 = 435

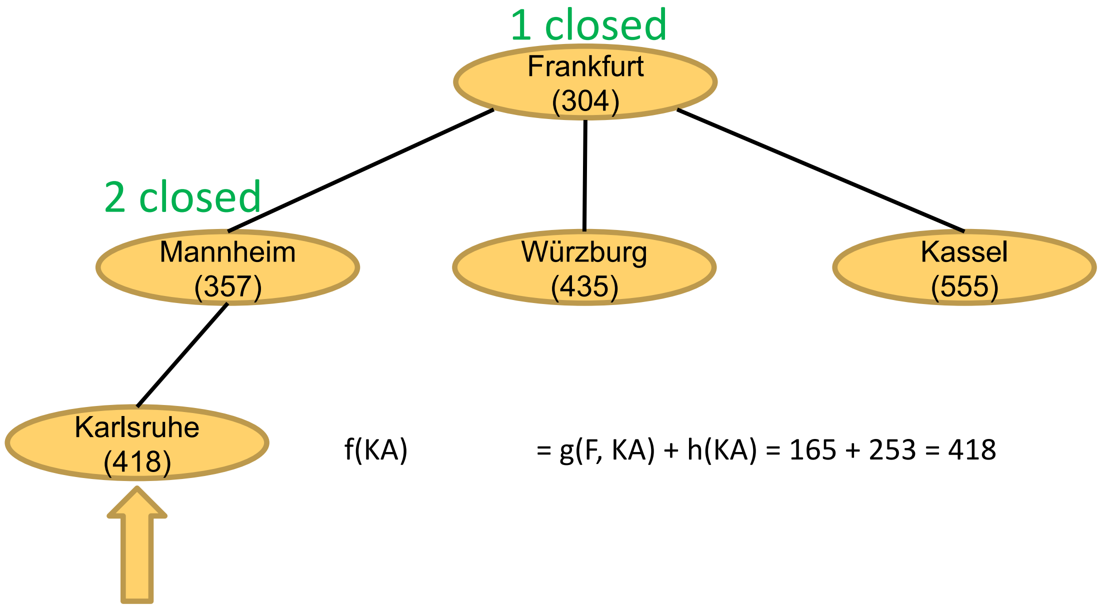
f(KS) = g(F, KS) + h(KS) = 173 + 382 = 555

A* Search – Example

Shortest path from Frankfurt to München?



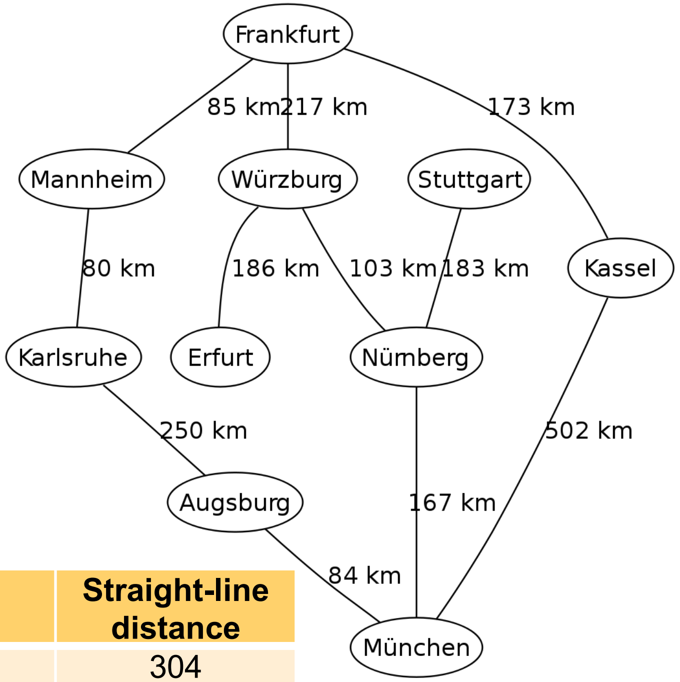
City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0



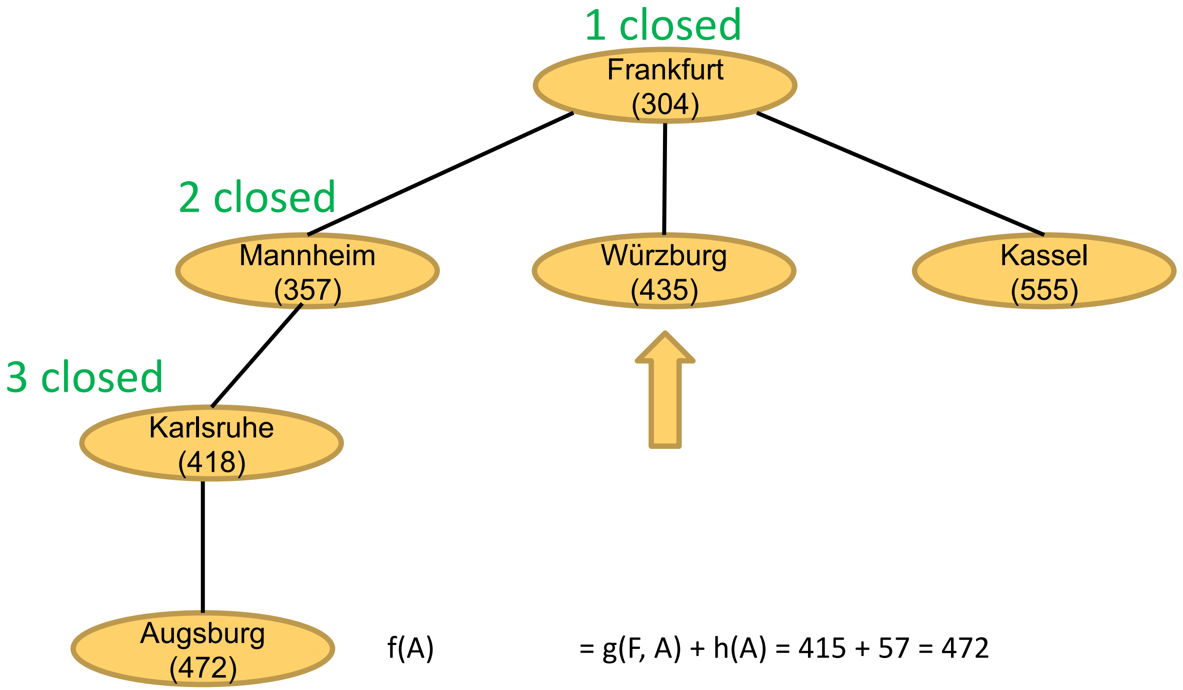
f(KA) = g(F, KA) + h(KA) = 165 + 253 = 418

A* Search – Example

Shortest path from Frankfurt to München?



City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0

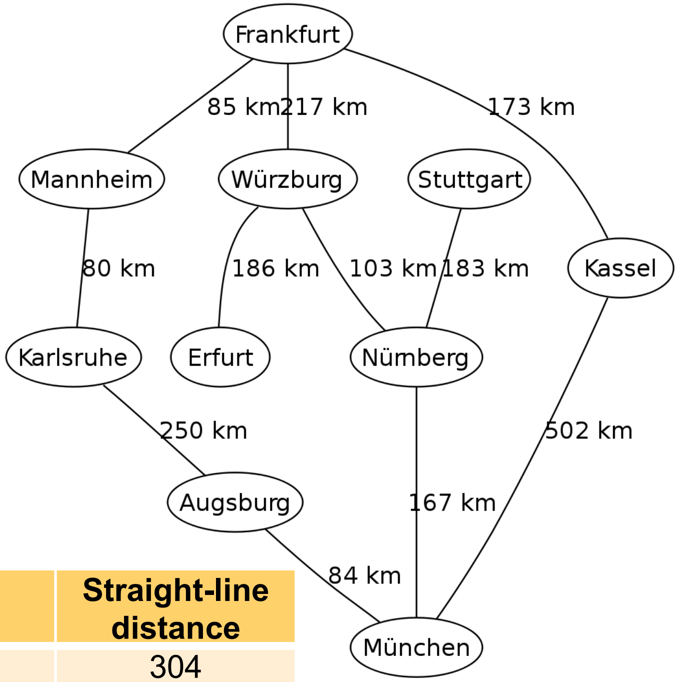


f(A) = g(F, A) + h(A) = 415 + 57 = 472

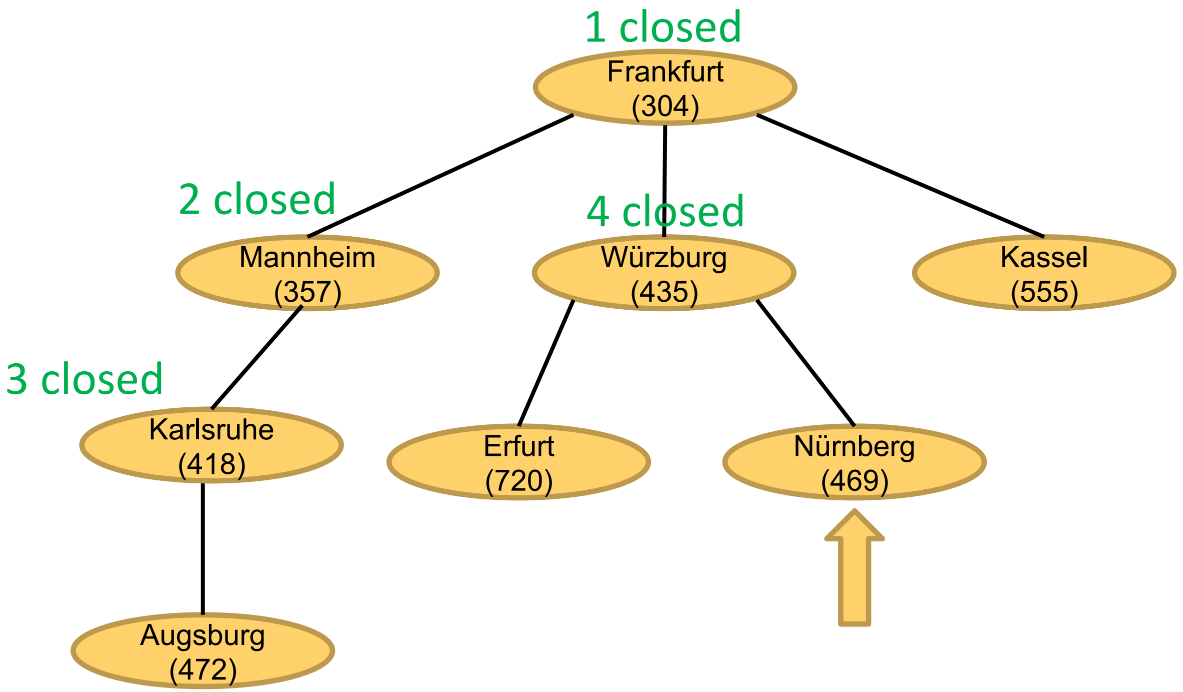
A* Search – Example

Technische Hochschule Rosenheim

Shortest path from Frankfurt to München?



City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0

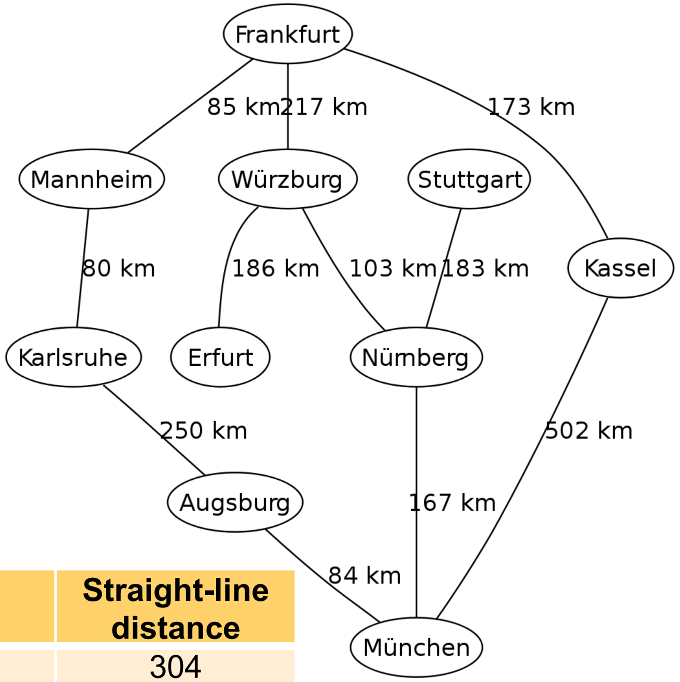


f(EF) = g(F, EF) + h(EF) = 403 + 317 = 720

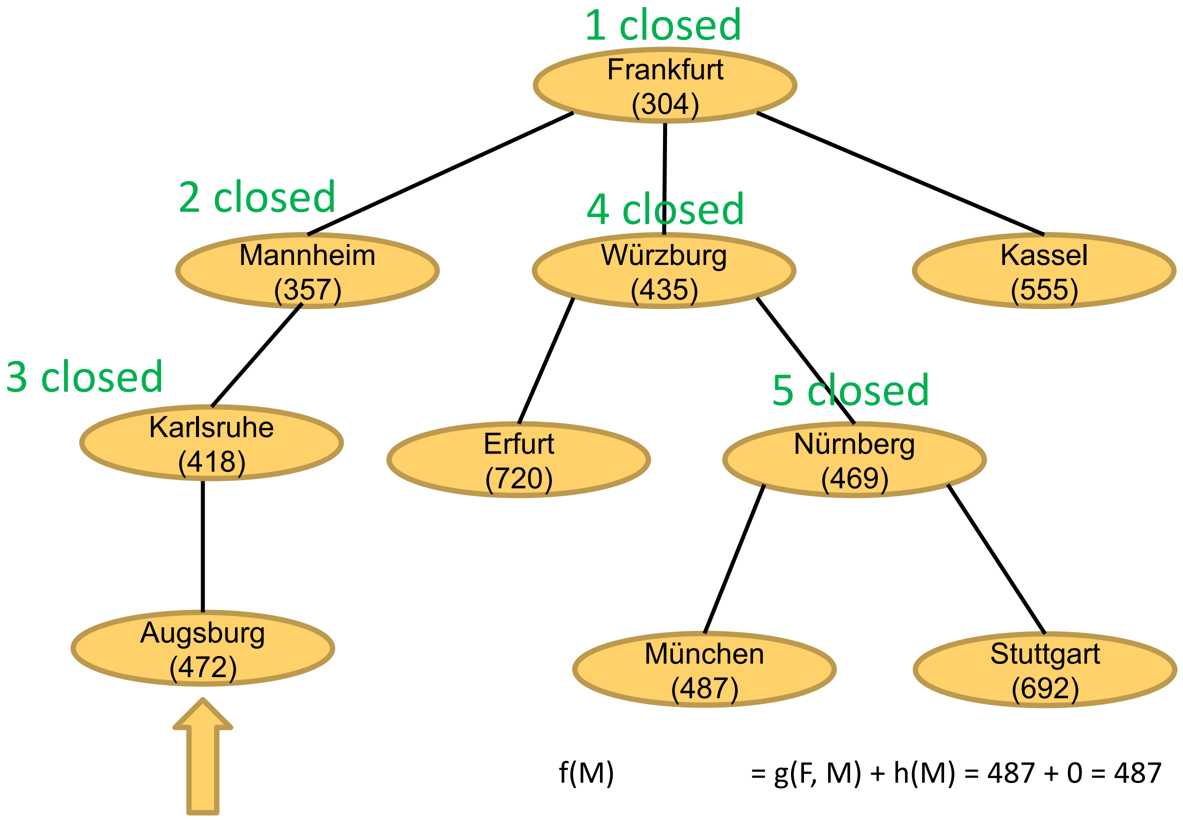
f(N) = g(F, N) + h(N) = 320 + 149 = 469

A* Search – Example

Shortest path from Frankfurt to München?



City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0

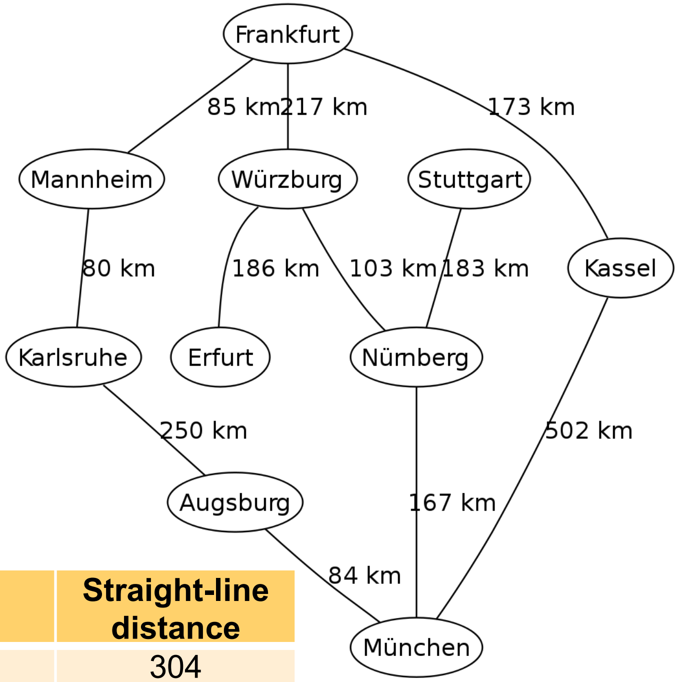


f(M) = g(F, M) + h(M) = 487 + 0 = 487

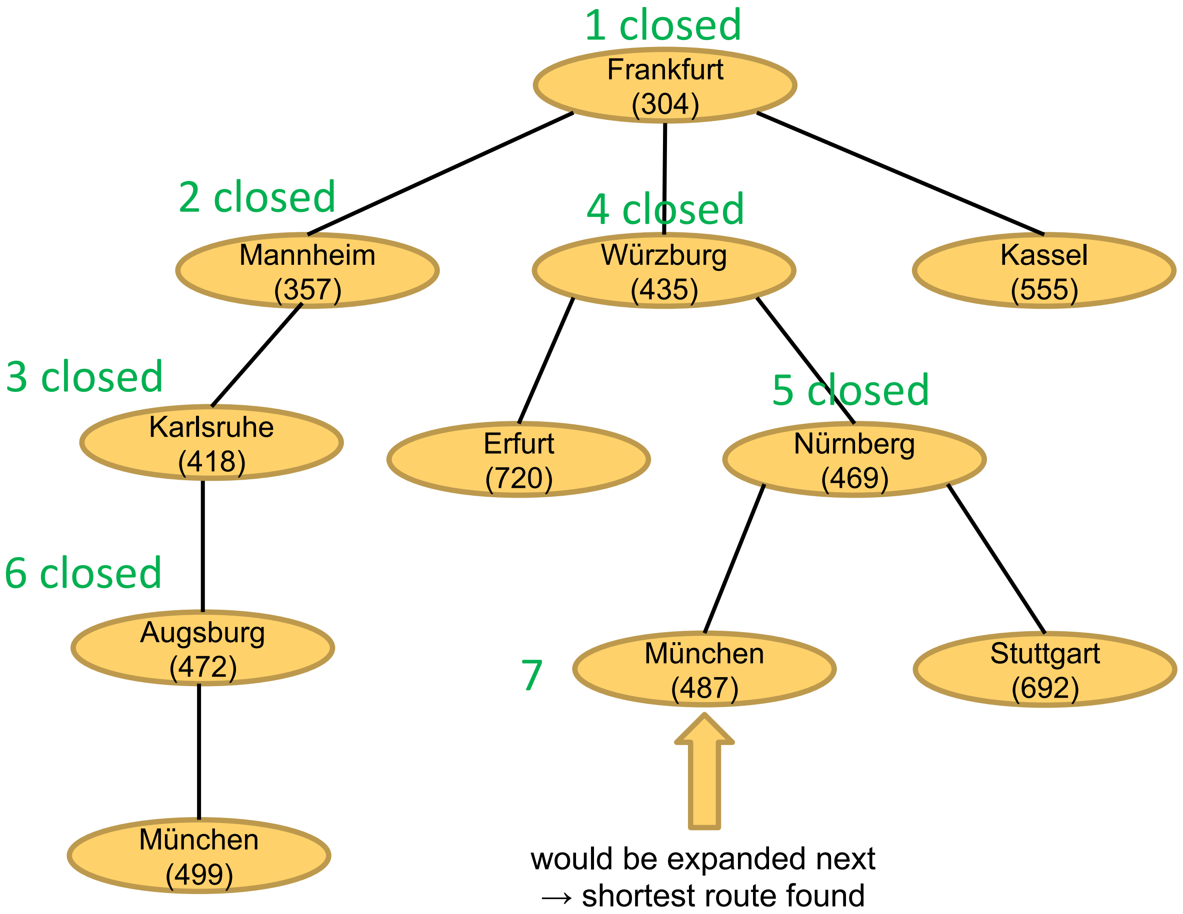
f(S) = g(F, S) + h(M) = 503 + 189 = 692

A* Search – Example

Shortest path from Frankfurt to München?



City	Straight-line distance
Frankfurt (F)	304
Mannheim (MA)	272
Würzburg (WÜ)	218
Stuttgart (S)	189
Kassel (KS)	382
Karlsruhe (KA)	253
Erfurt (EF)	317
Nürnberg (N)	149
Augsburg (A)	57
München (M)	0



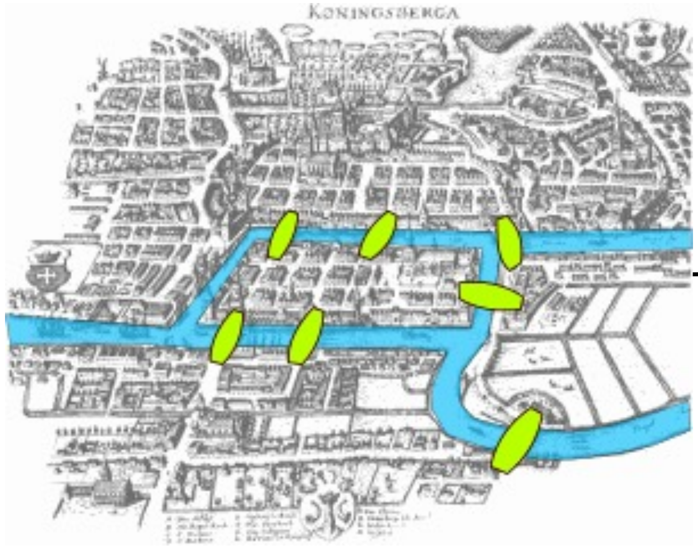
A* Search – Evaluation

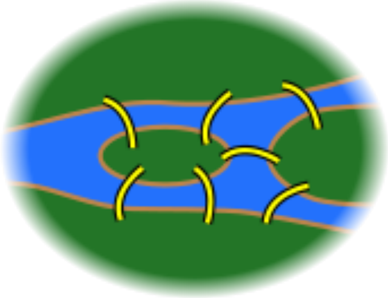
- Completeness: YES
- Optimality:
  - YES, if constraints on $h(n)$ are met
  - A* is even optimal-efficient: It can be proofed that there exists no other optimal algorithm expanding fewer nodes (exception: same rating of different nodes, then the selection is random)
- Time complexity: exponential in the length of the shortest path
- Space complexity: all nodes have to be kept in memory
  - therefore, memory consumption is the main problem, not computation time
  - there are variants of A* like IDA* (Iterative Deepening A*) addressing this problem, typically leading to higher computation times though

Back to the Start

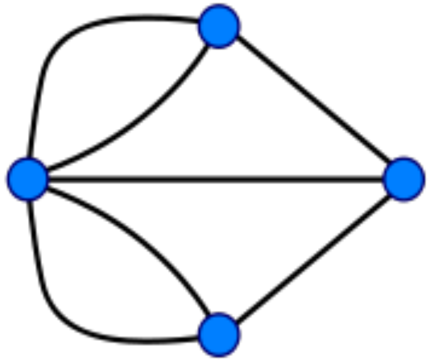
Euler 1736:

Is there a circular route through Königsberg that crosses each of the seven bridges over the Pregel exactly once?





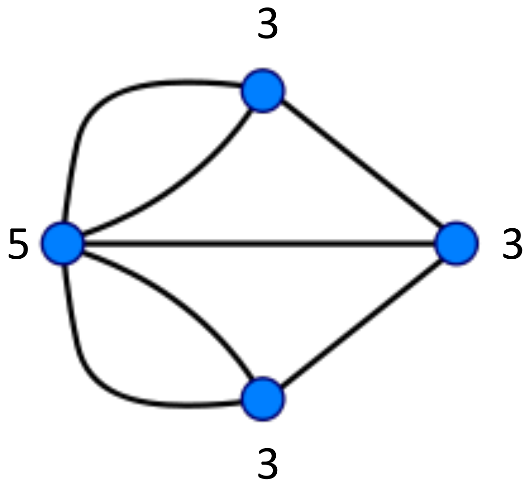
Is there a circular route that contains each edge exactly once?



Eulerian Graphs

- Eulerian cycle (*Eulerkreis*): closed trail that contains all edges of the graph exactly once
- Eulerian graph (*Eulergraph*): connected graph that contains an Eulerian trail

- undirected graphs: Graph is Eulerian if and only if the graph is connected and the degree of each node is even



Is there a circular route that contains each edge exactly once?

NO

Eulerian Graphs

- The decision problem is therefore relatively easy to solve
- Is a given graph Eulerian?
  Check whether graph is connected and determine degree of each node

- There are efficient algorithms for actually finding an Eulerian cycle
  - will not be considered here

Hamiltonian Graphs

- Is there a closed path that contains each node exactly once?
- Special case: Traveling Salesman Problem
  - look for the shortest Hamilton cycle
- This problem is much more difficult
  - NP-complete (⟶ Theoretical Computer Science 2nd Sem.)
  - except in some special cases
    - e.g., any complete graph with 3 nodes or more is Hamiltonian