Introduction to Artificial Intelligence

State Space with Costs on Actions, Heuristic Search

Ing. Tomas Borovicka

Department of Theoretical Computer Science (KTI), Faculty of Information Technology (FIT)
Czech Technical University in Prague (CVUT)

BIE-ZUM, LS 2014/15, 3. lecture

https://edux.fit.cvut.cz/courses/BIE-ZUM/

Summary of Previous Lecture

- We learnt how to define a problem using a state space.
- We learnt some basics from the Graph Theory.
- We learnt how to search the state space without any additional information about the problem.
- Uninformed Search Algorithms:
  - Breadth-First Search (BFS)
    - ★ Expands nodes from the initial node level by level.
    - ★ Complete and optimal.
    - ★ Exponential time and space complexity.*
  - Depth-First Search (DFS)
    - ★ Expands the deepest from the opened nodes and directly travels to the deepest level of the search tree.
    - ★ Incomplete and sub-optimal.
    - ★ Exponential time complexity, but linear space complexity.*

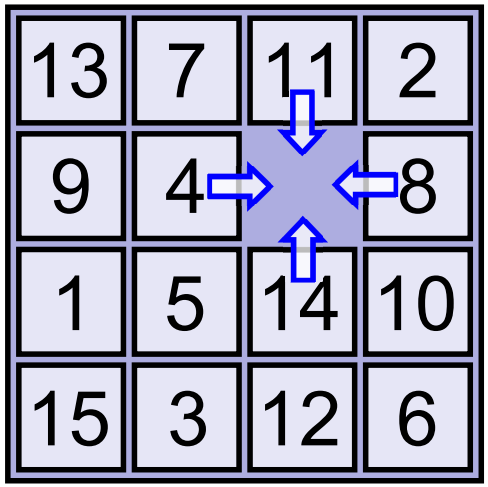
* in terms of tree search with the branching factor b and the depth d.

Costs on Actions

- Number of actions (moves) in a solution sequence is not always relevant metric for quality of a solution.
- Actions (state transitions) can differ in cost and our metric can be the total cost of a solution.
- In such case instead of sequence length we are interested in the path cost.

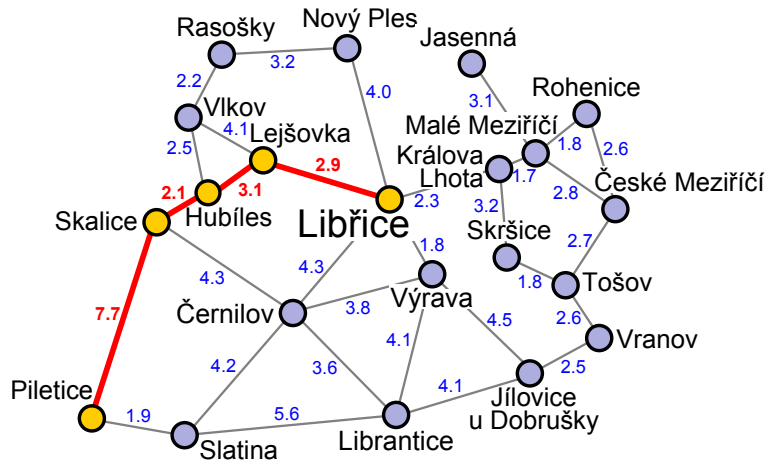
Sliding-block Puzzles

- Number of slides (actions).



Route Finding

- Length of the path (km).



State Space with Costs on Actions

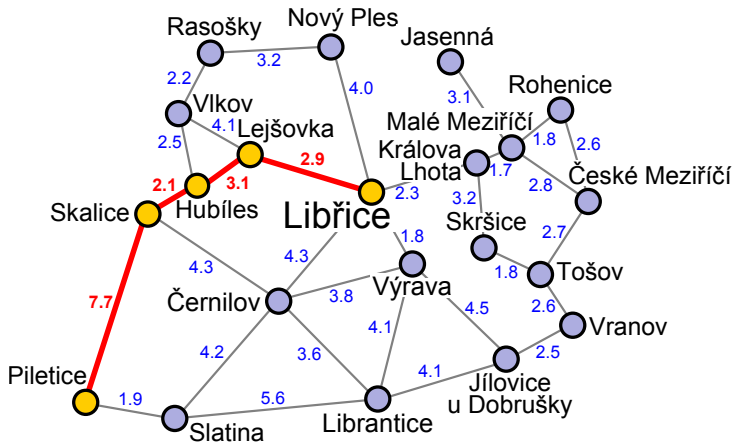
- With each action is associated a **non-negative cost**.

Definition (The State Space with Costs on Actions)

A state-space with costs on actions is a tuple $\langle S, A, \mathcal{I}, G, c \rangle$, such as (S, A) is a directed graph with a set of nodes S representing states and a set of edges A representing actions, $\mathcal{I}$ is a set of initial states, G is a set of goal states and $c(a)$ is a cost function assigning a non negative cost to each action a.

The Cost of a Solution

- The cost of a solution is a sum of all the costs along the path i.e. the cost of the path.



$C(p) = 7.7 + 2.1 + 3.1 + 2.9 =$ **15.8**

Definition (The cost of a path)

Let $\langle S, A, \mathcal{I}, G, c \rangle$ be a state space with cost function c and let $p = (s_1, a_1, s_2, \ldots, a_{n-1}, s_n)$ be an oriented path from the state s_1 to the state s_2. The path cost is then a sum of costs of all actions along the path:

$$C(p) = \sum_{i=1}^{n-1} c(a_i).$$

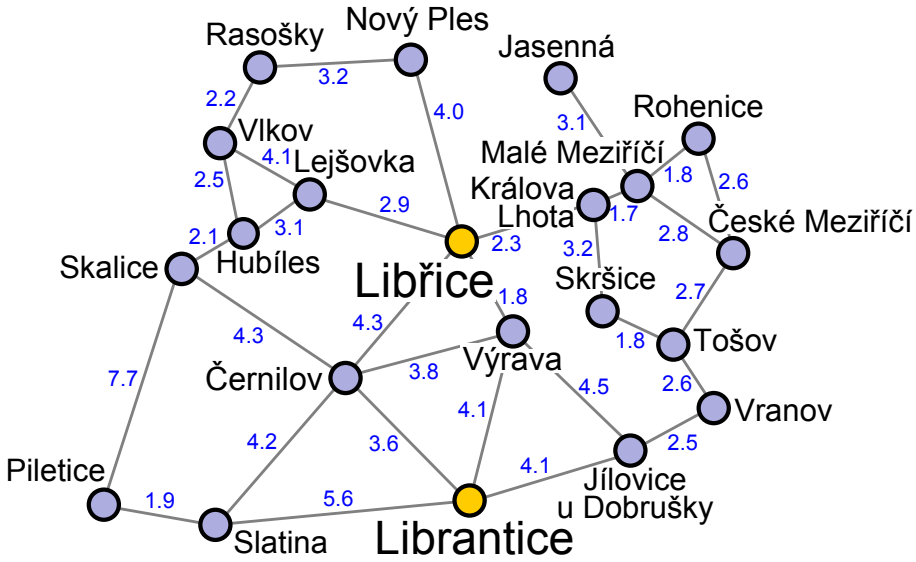
Dijkstra's Algorithm

- An algorithm for finding the shortest path between two vertices in a graph.
- Actually, it finds the shortest paths
  - from a source node s_i to a goal node s_n,
  - from a source node s_i to all other nodes in the graph.
- For each node the total path cost from the initial state to the given node is remembered.
- The node with the shortest path (lowest cost) is expanded.
- If a shorter path for already opened node is found, path cost to the node and node's predecessor is updated (i.e. the node is moved to appropriate branch in the search tree).
- Implemented using min-priority queue.

Complete: Yes
Optimal: Yes
Time: $O(b^d)$
Space: $O(b^d)$

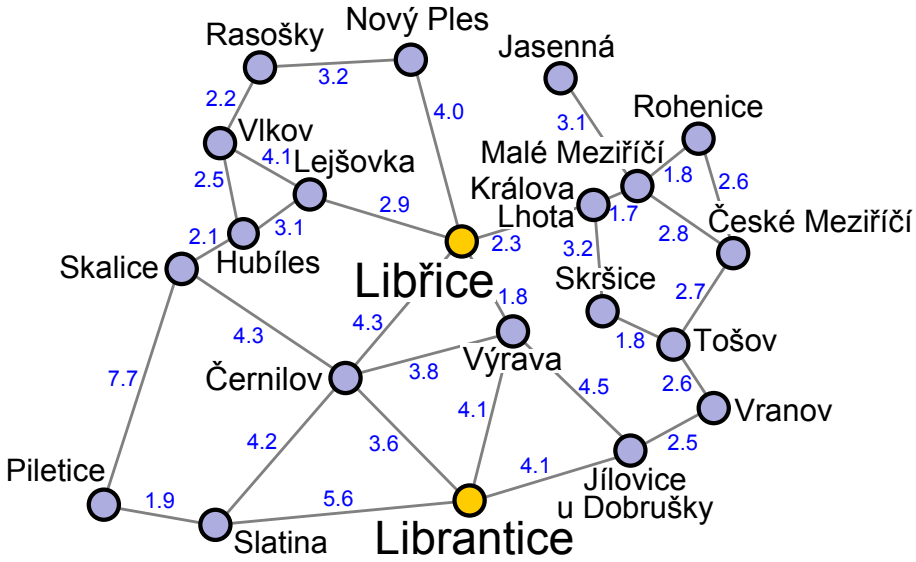
Dijkstra's Algorithm : Example

The shortest path from Librantice to Libřice?



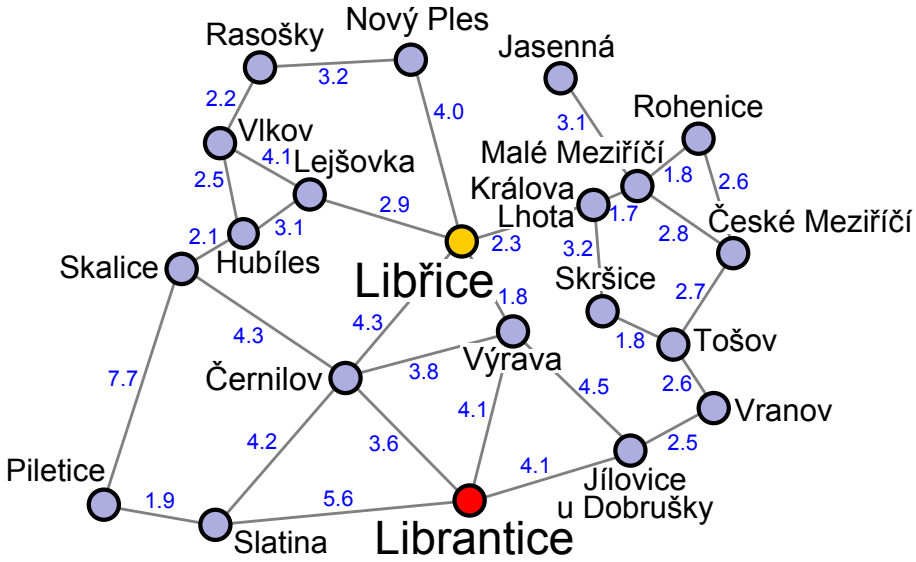
Dijkstra's Algorithm : Example

The shortest path from Librantice to Libřice?



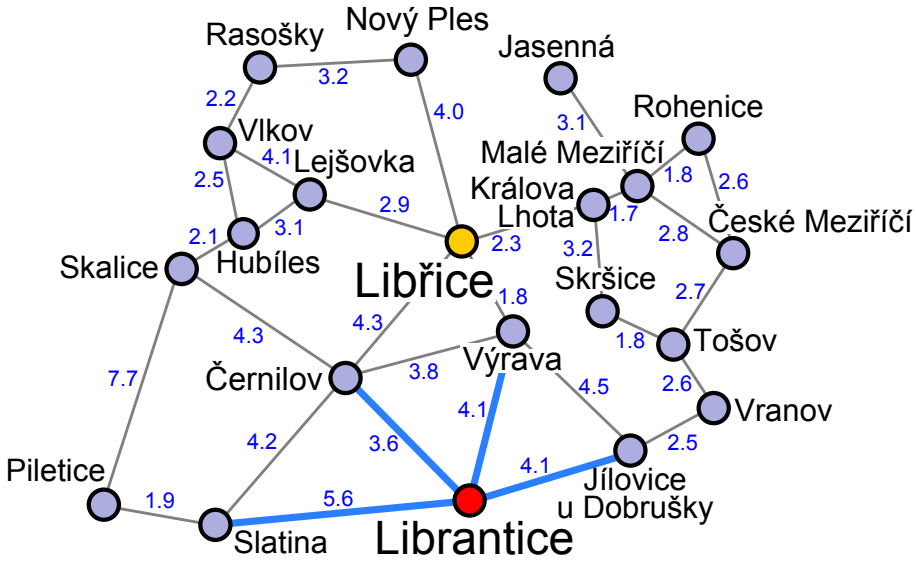
Dijkstra's Algorithm : Example

The shortest path from Librantice to Libřice?



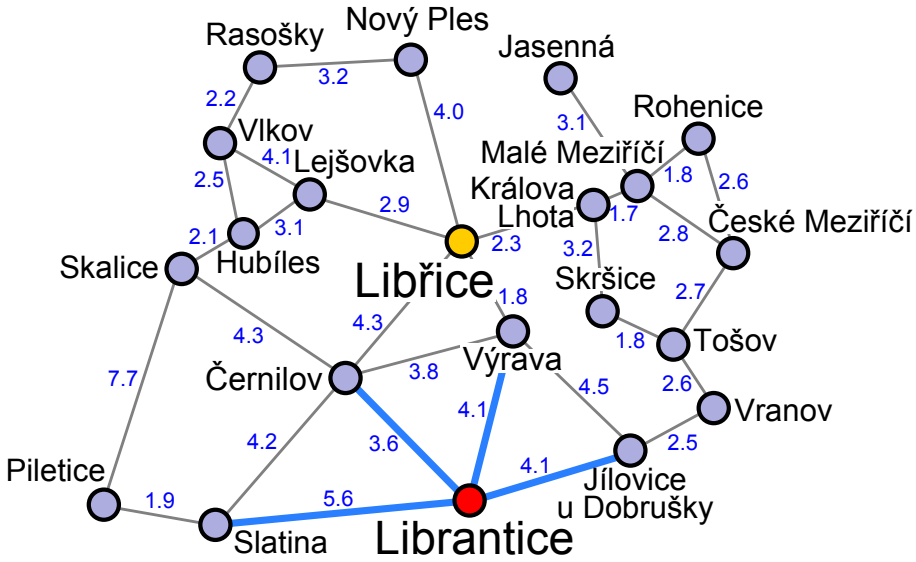
Dijkstra's Algorithm : Example

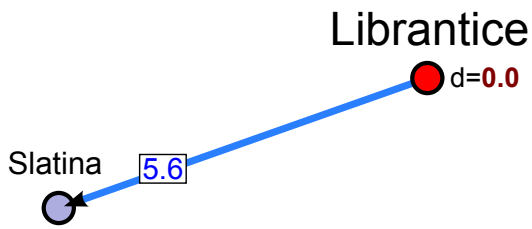
The shortest path from Librantice to Libřice?



Dijkstra's Algorithm : Example

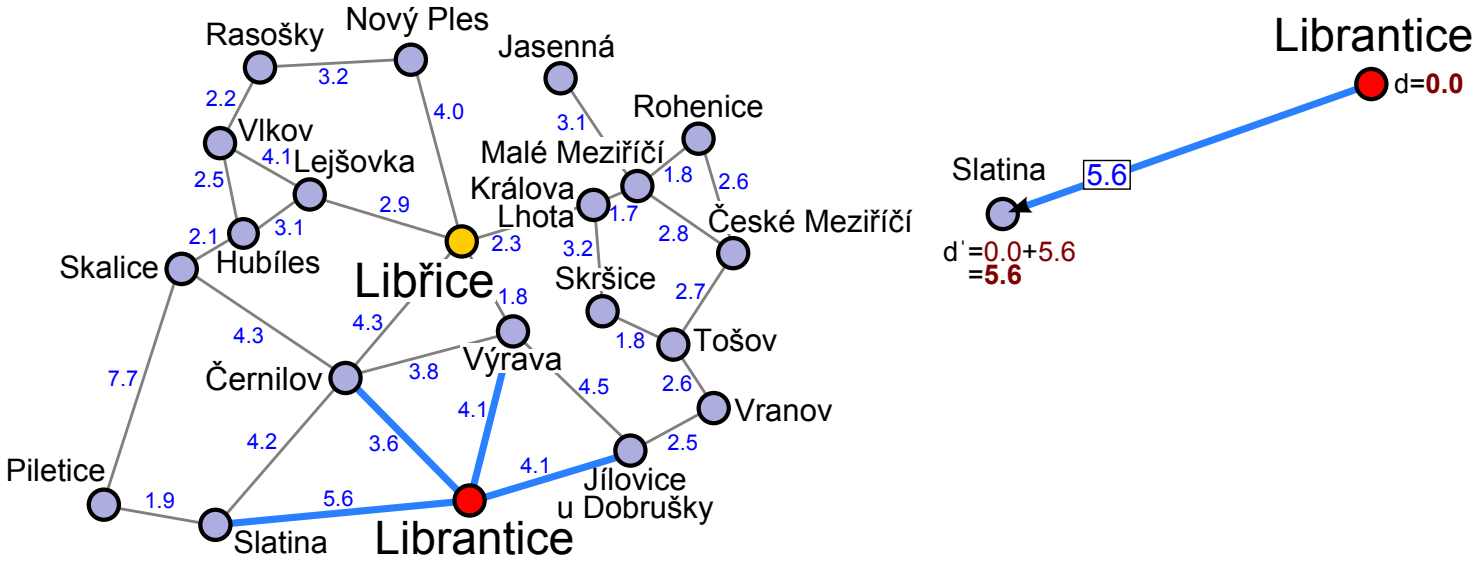
The shortest path from Librantice to Libřice?





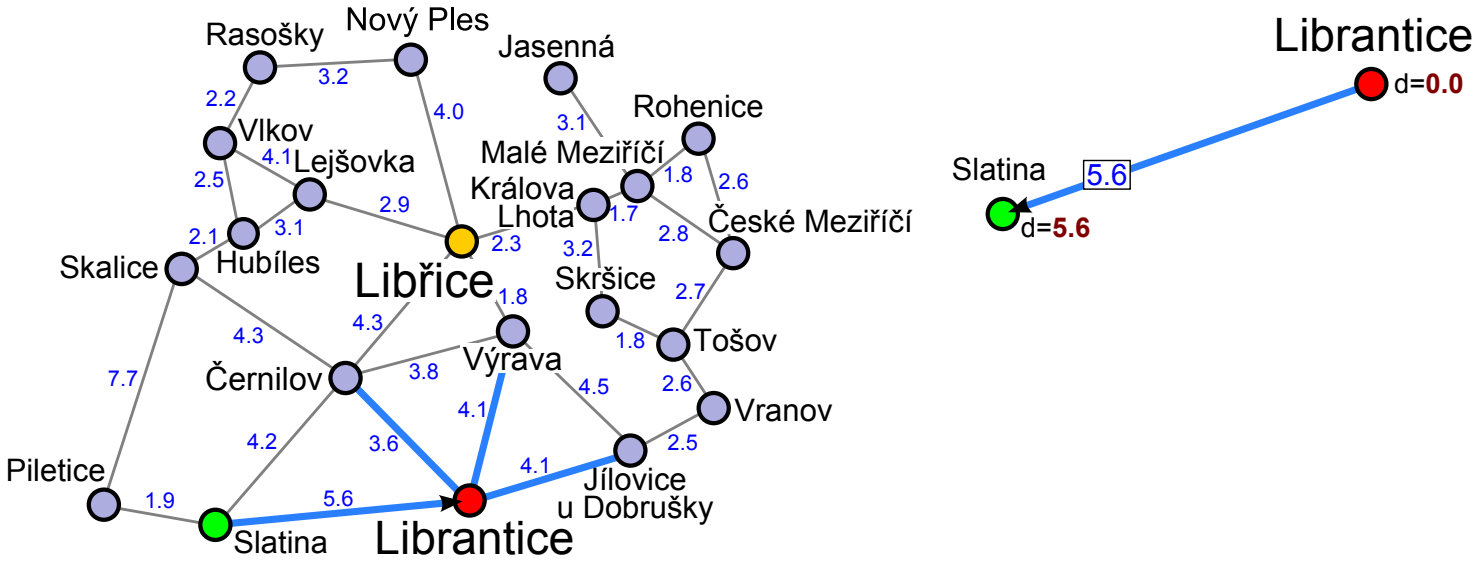
Dijkstra's Algorithm : Example

The shortest path from Librantice to Libřice?



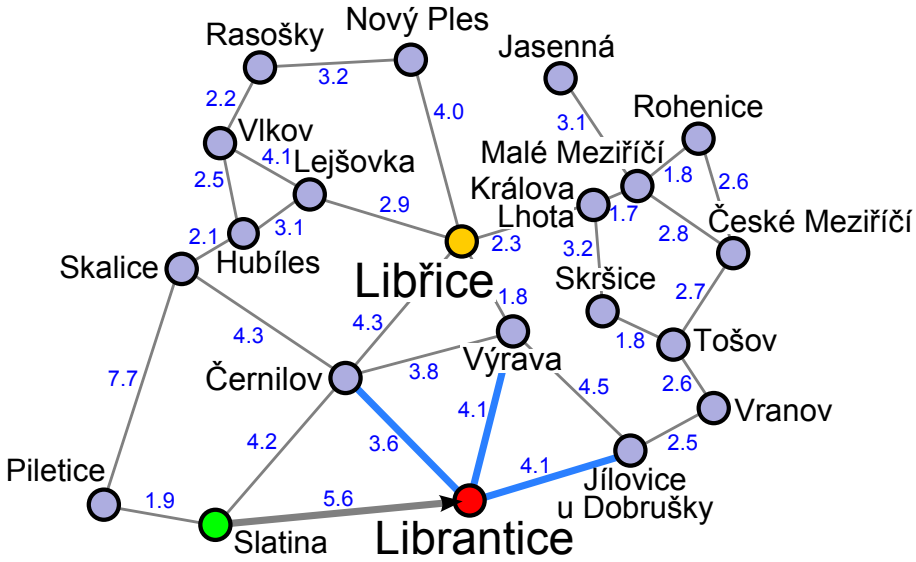
Dijkstra's Algorithm : Example

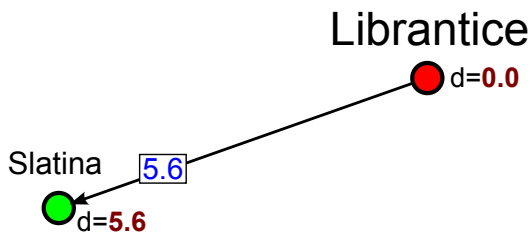
The shortest path from Librantice to Libřice?



Dijkstra's Algorithm : Example

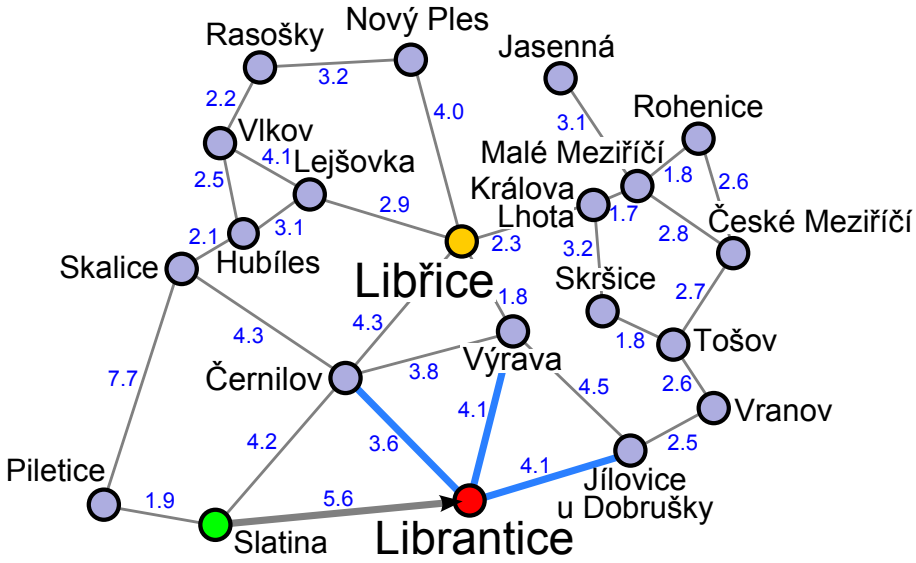
The shortest path from Librantice to Libřice?

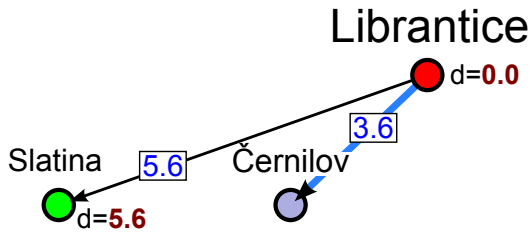




Dijkstra's Algorithm : Example

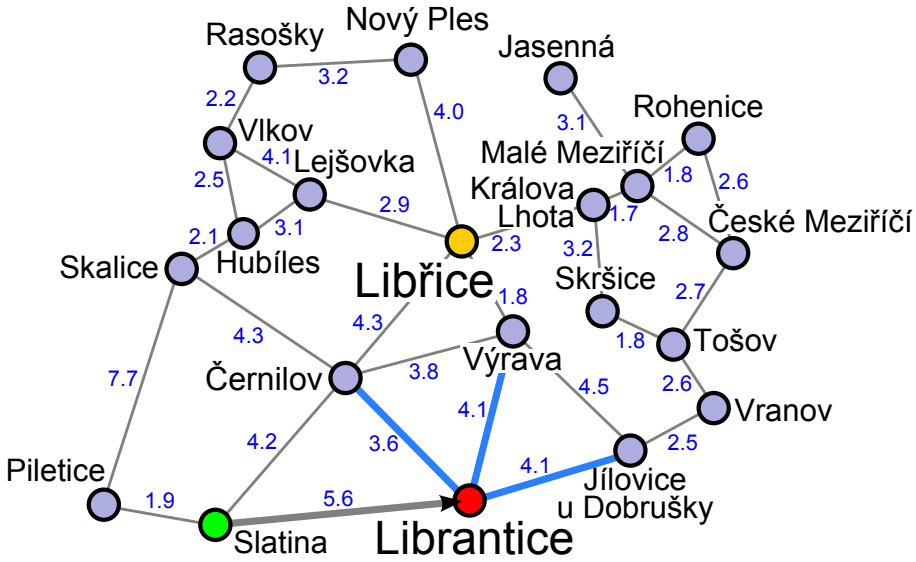
The shortest path from Librantice to Libřice?

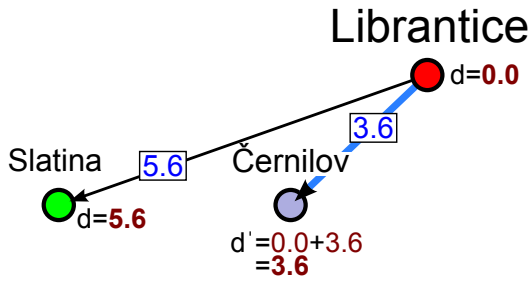




Dijkstra's Algorithm : Example

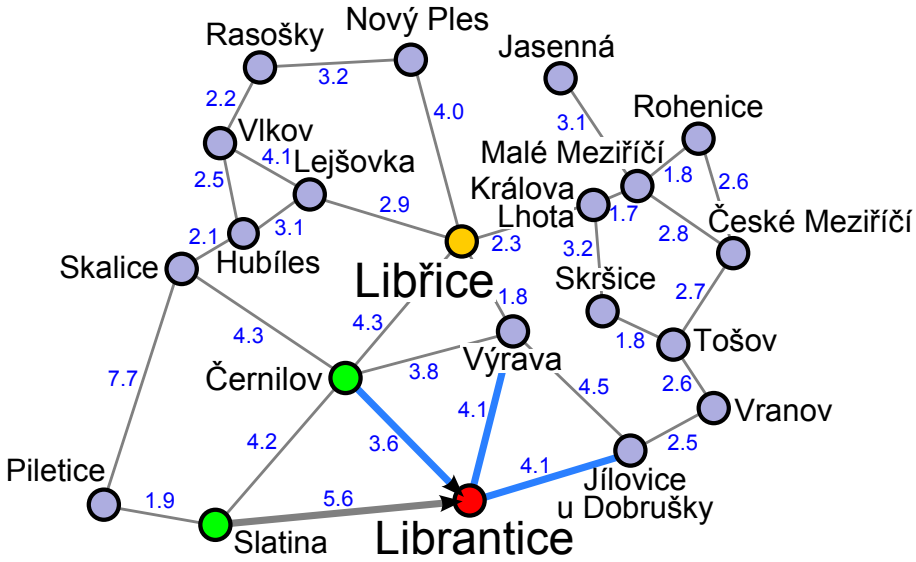
The shortest path from Librantice to Libřice?

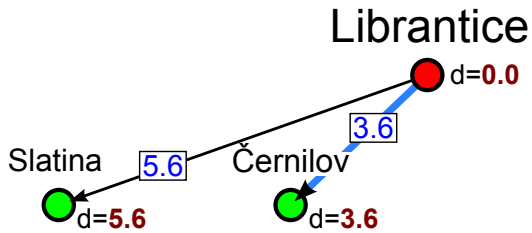




Dijkstra's Algorithm : Example

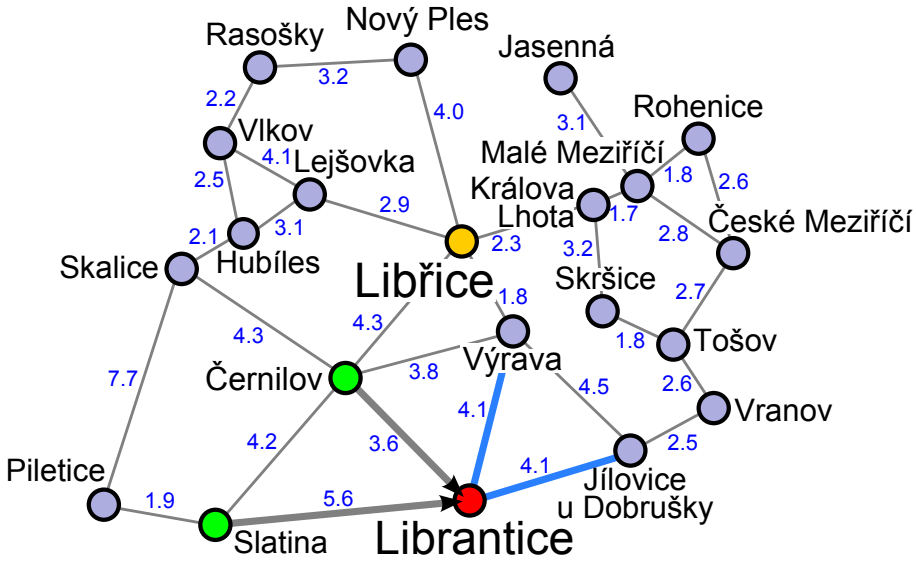
The shortest path from Librantice to Libřice?

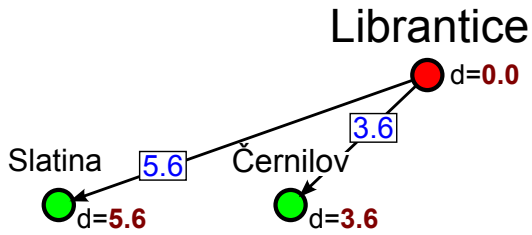




Dijkstra's Algorithm : Example

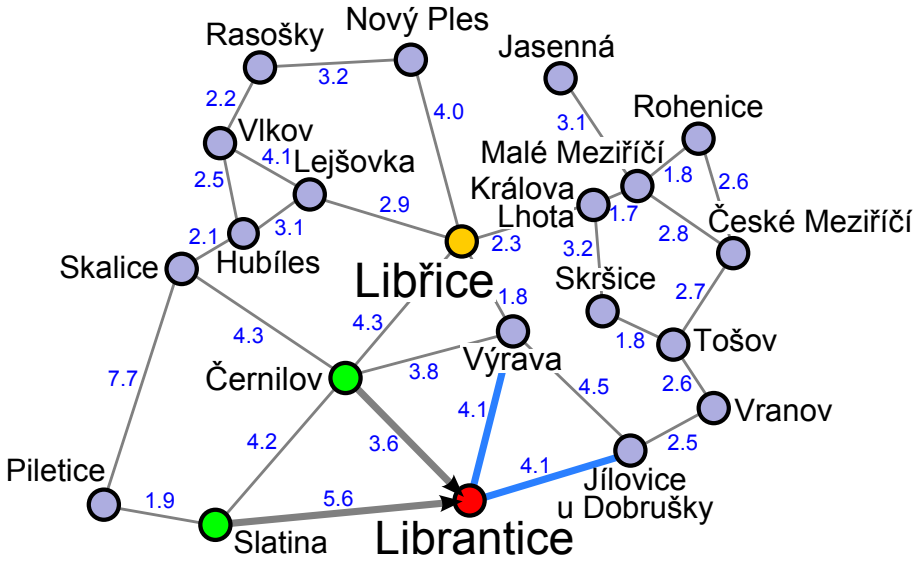
The shortest path from Librantice to Libřice?

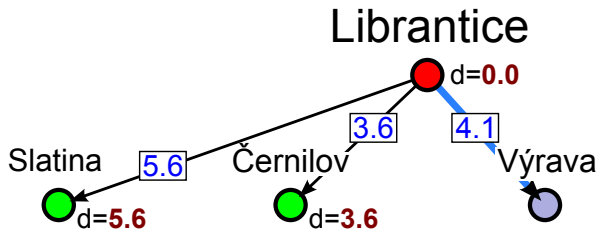




Dijkstra's Algorithm : Example

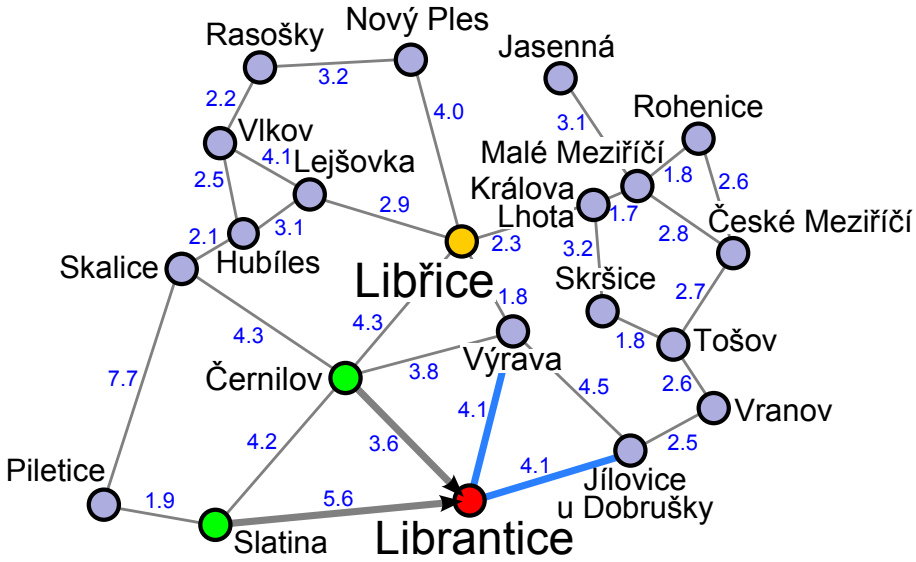
The shortest path from Librantice to Libřice?

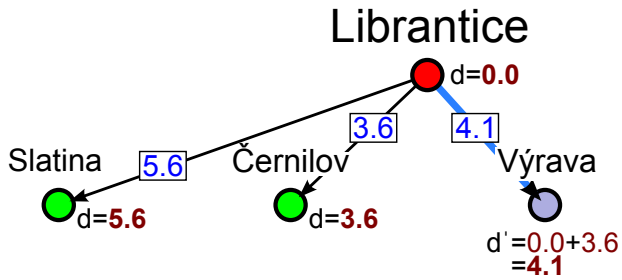




Dijkstra's Algorithm : Example

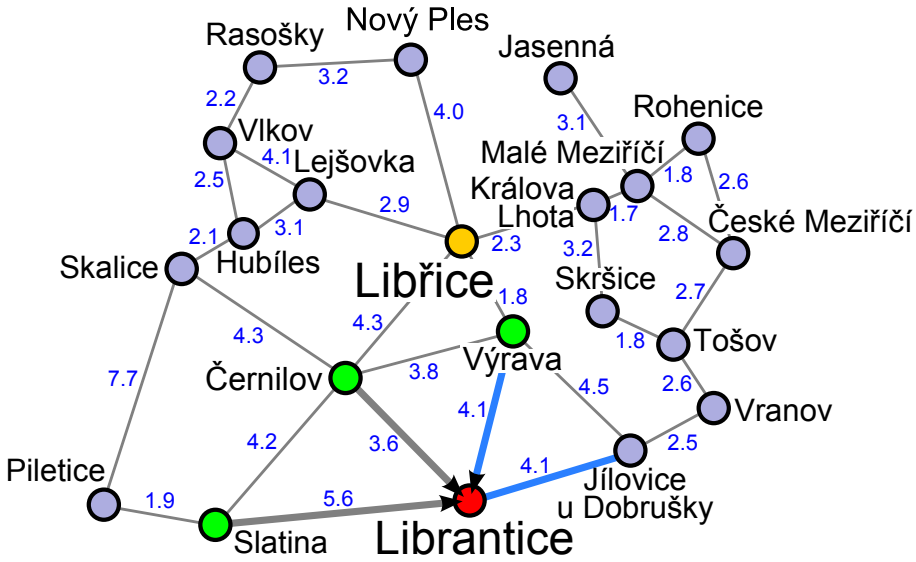
The shortest path from Librantice to Libřice?

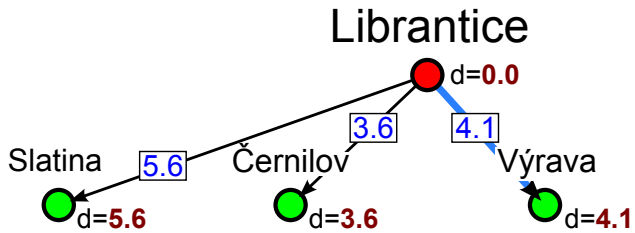




Dijkstra's Algorithm : Example

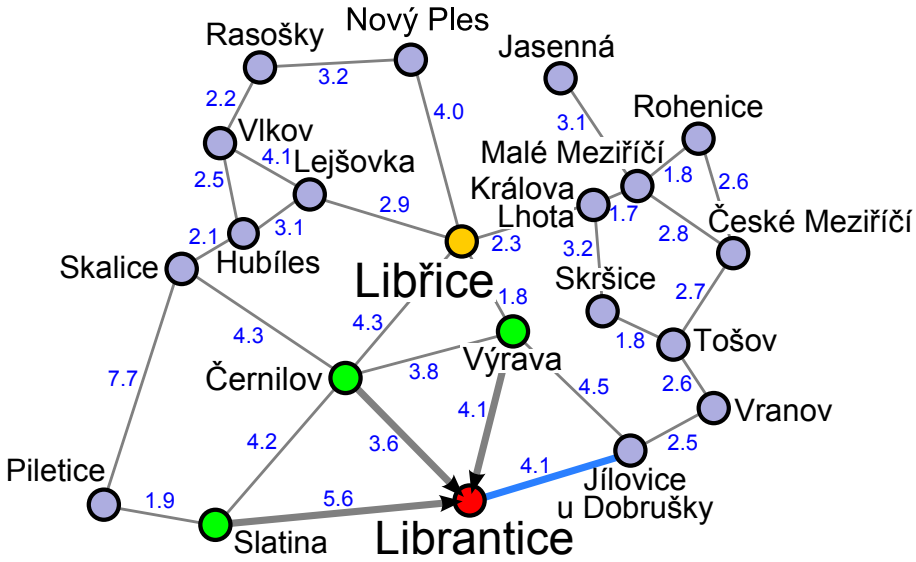
The shortest path from Librantice to Libřice?

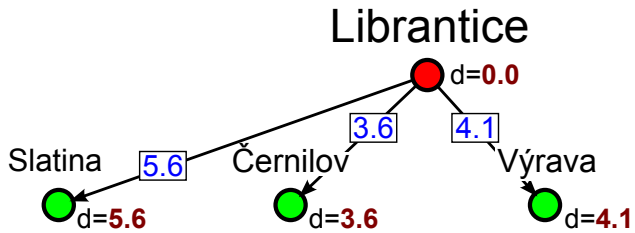




Dijkstra's Algorithm : Example

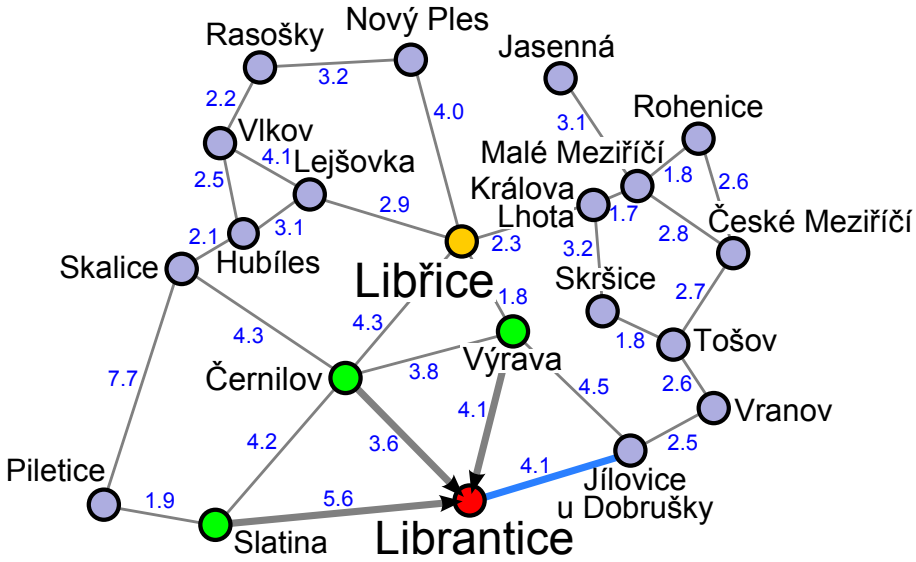
The shortest path from Librantice to Libřice?

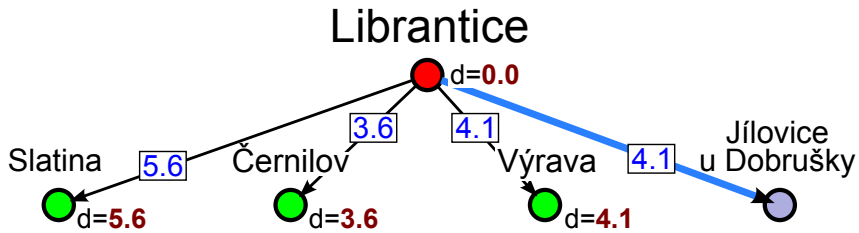




Dijkstra's Algorithm : Example

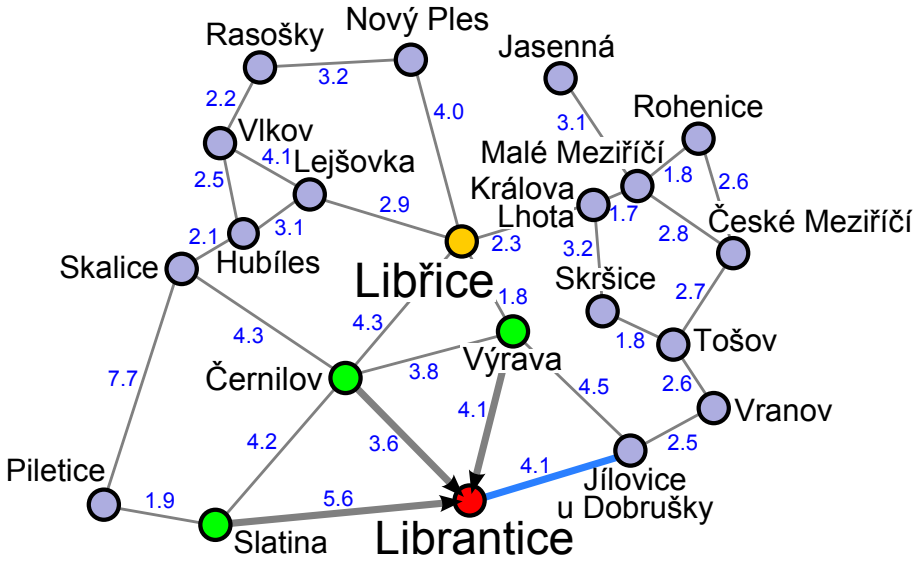
The shortest path from Librantice to Libřice?

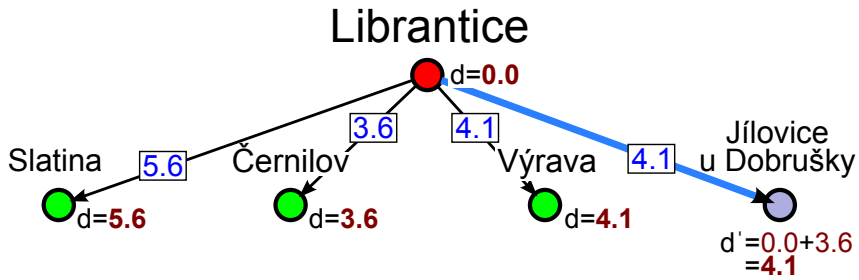




Dijkstra's Algorithm : Example

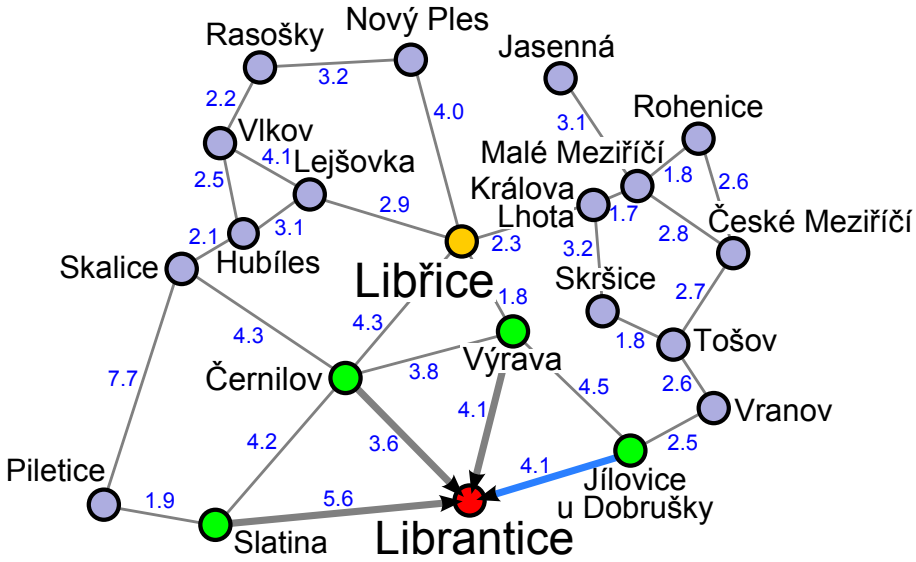
The shortest path from Librantice to Libřice?

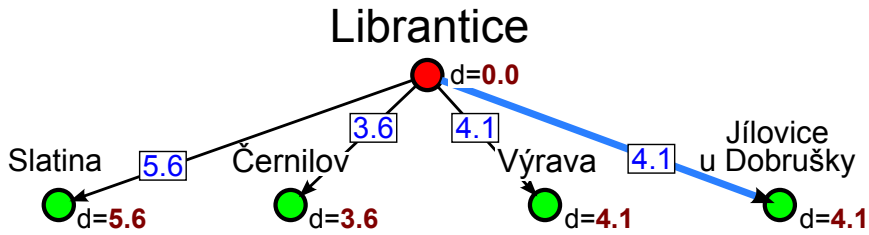




Dijkstra's Algorithm : Example

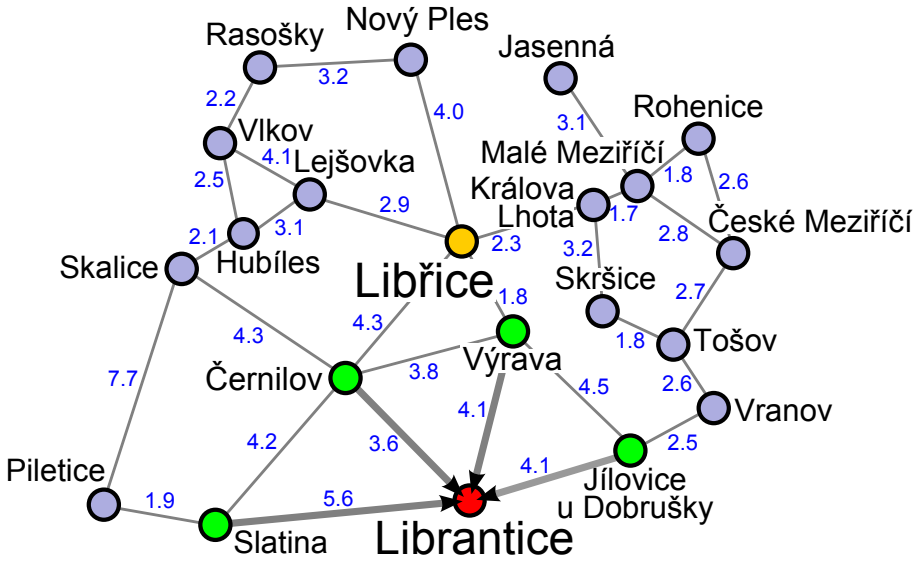
The shortest path from Librantice to Libřice?

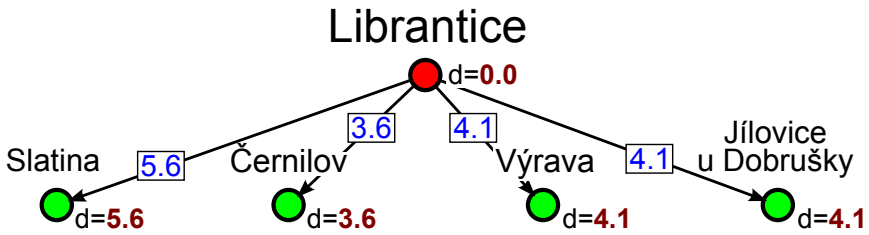




Dijkstra's Algorithm : Example

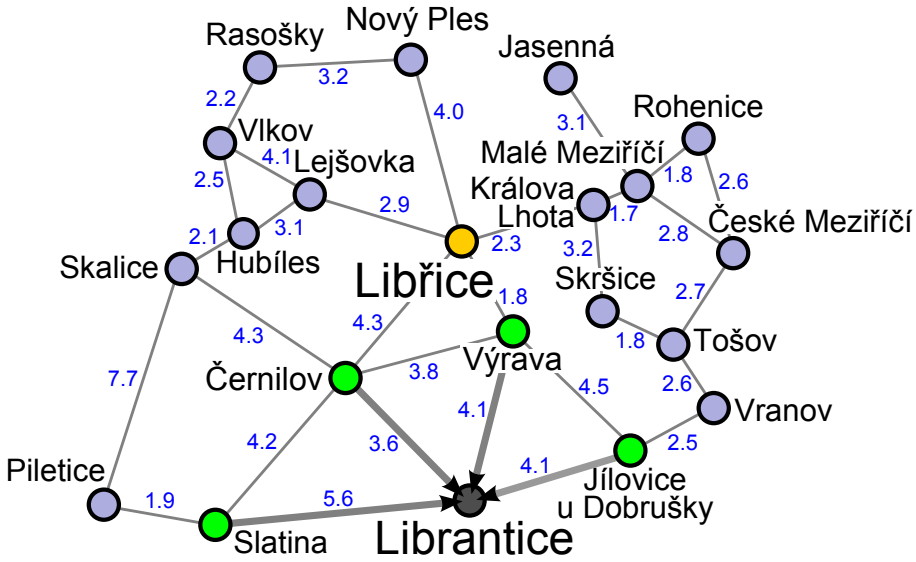
The shortest path from Librantice to Libřice?

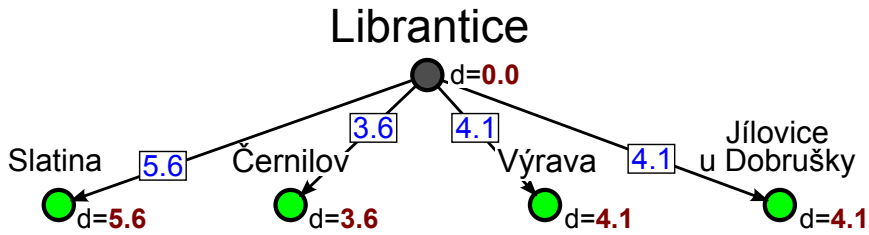




Dijkstra's Algorithm : Example

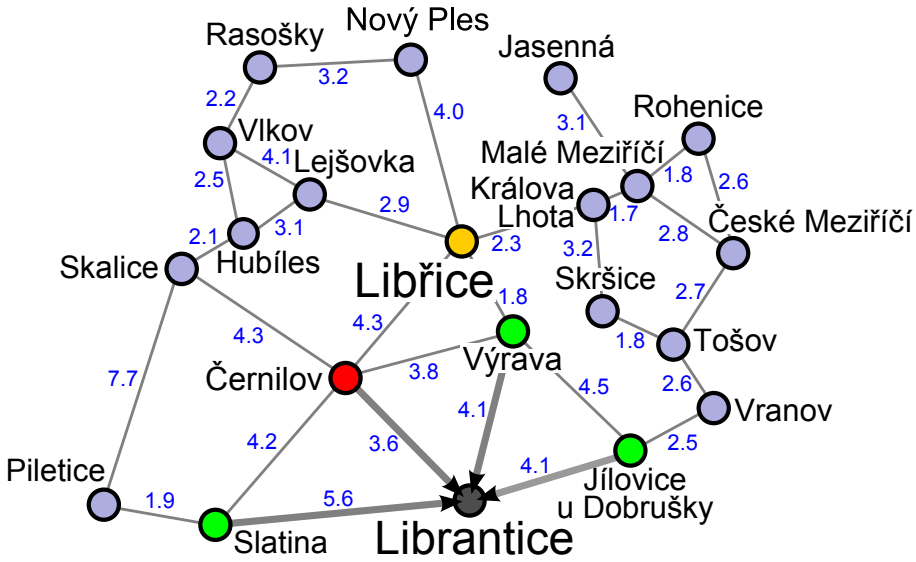
The shortest path from Librantice to Libřice?

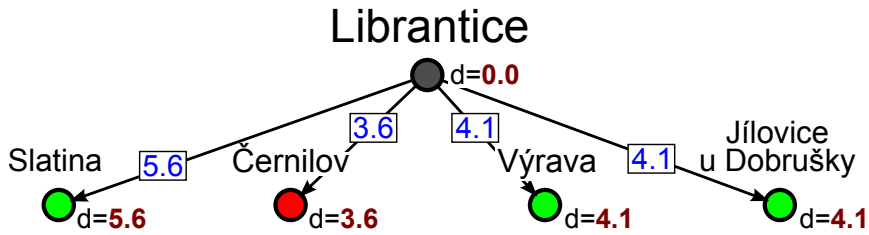




Dijkstra's Algorithm : Example

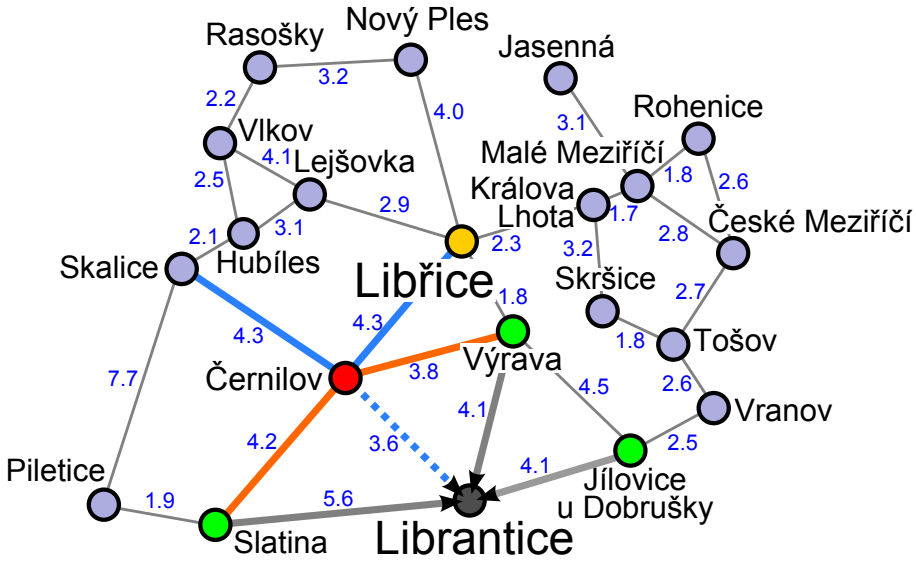
The shortest path from Librantice to Libřice?

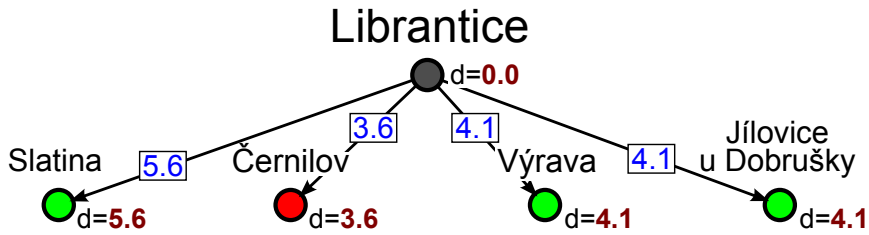




Dijkstra's Algorithm : Example

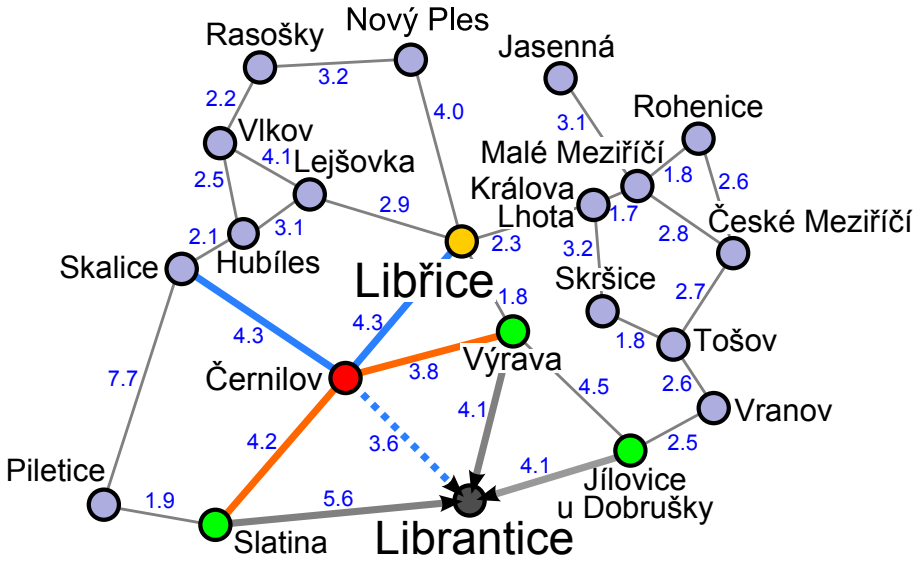
The shortest path from Librantice to Libřice?

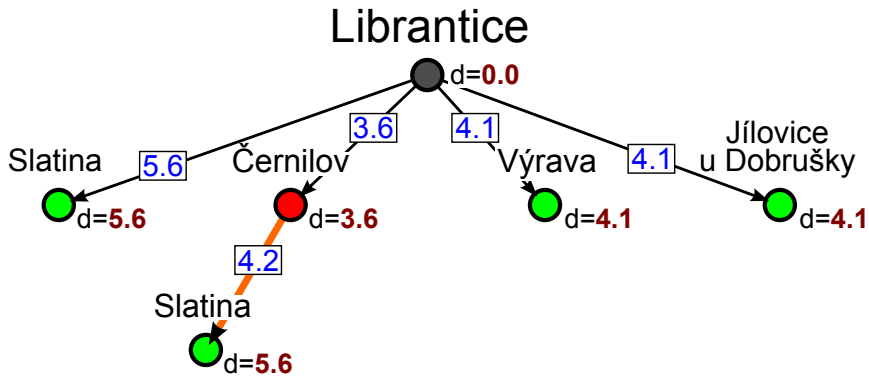




Dijkstra's Algorithm : Example

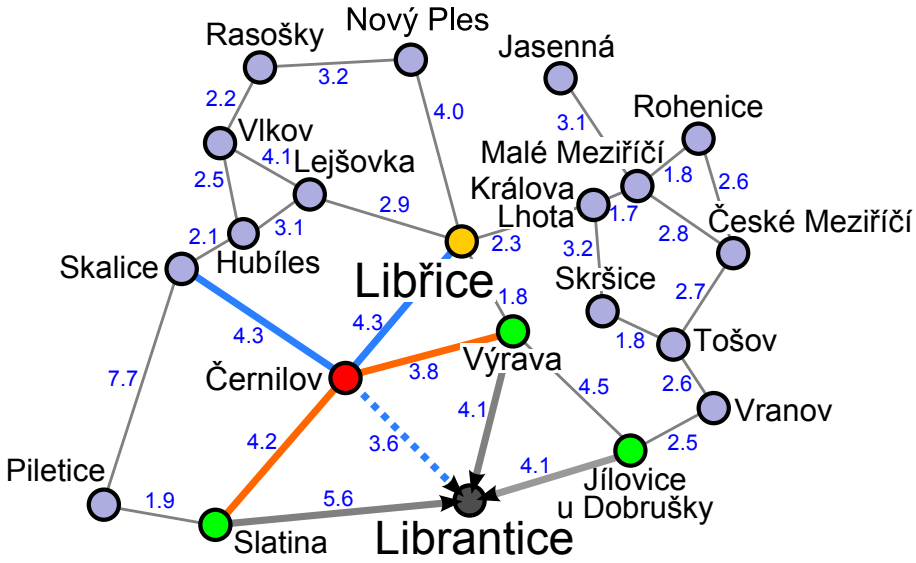
The shortest path from Librantice to Libřice?

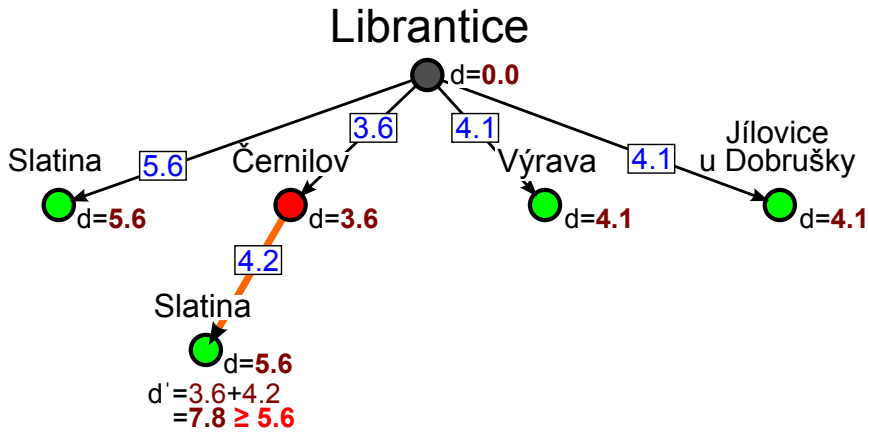




Dijkstra's Algorithm : Example

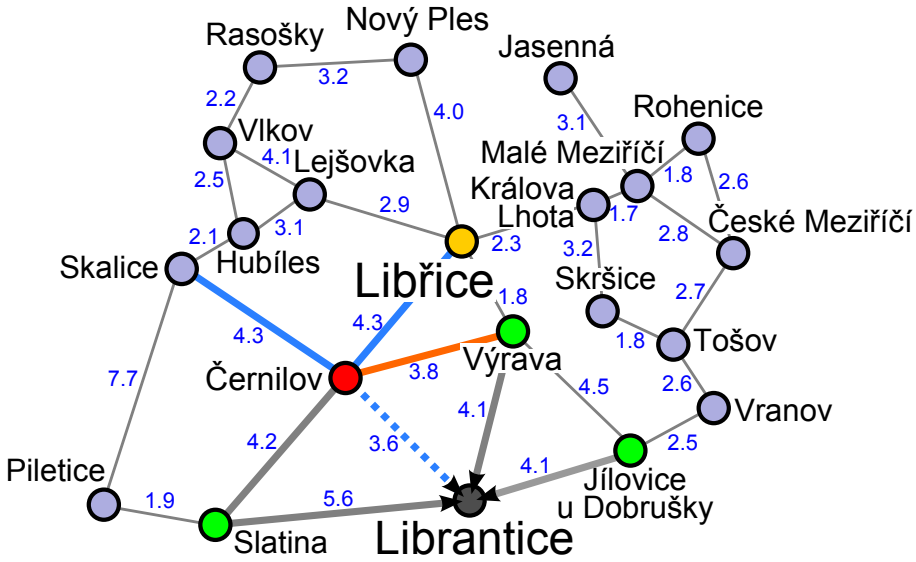
The shortest path from Librantice to Libřice?

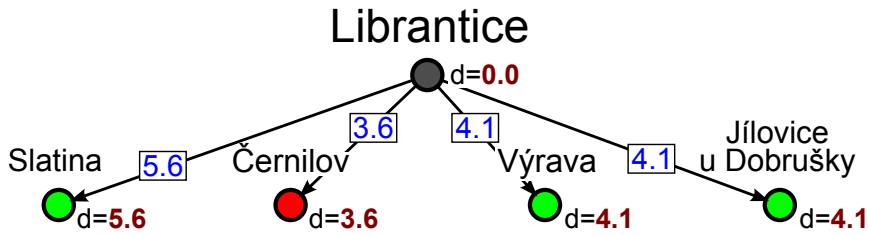




Dijkstra's Algorithm : Example

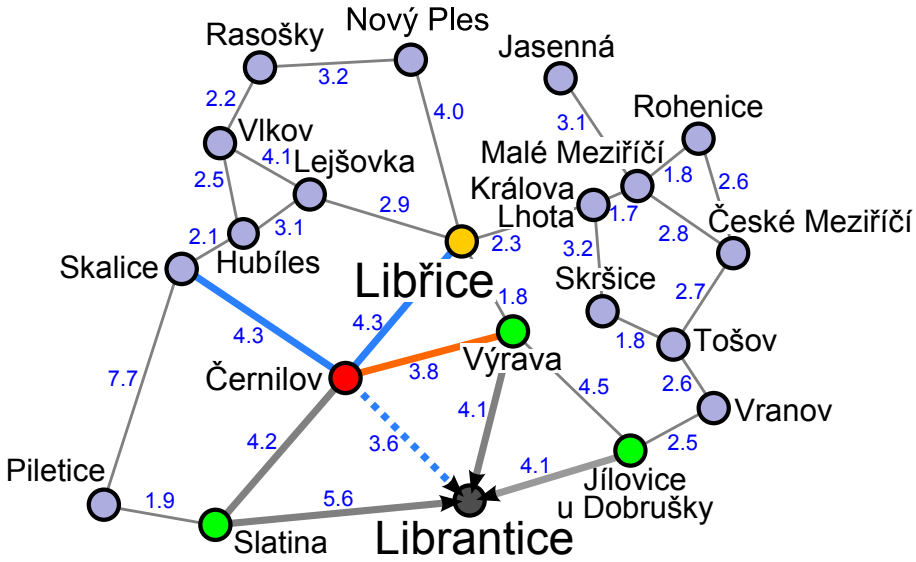
The shortest path from Librantice to Libřice?

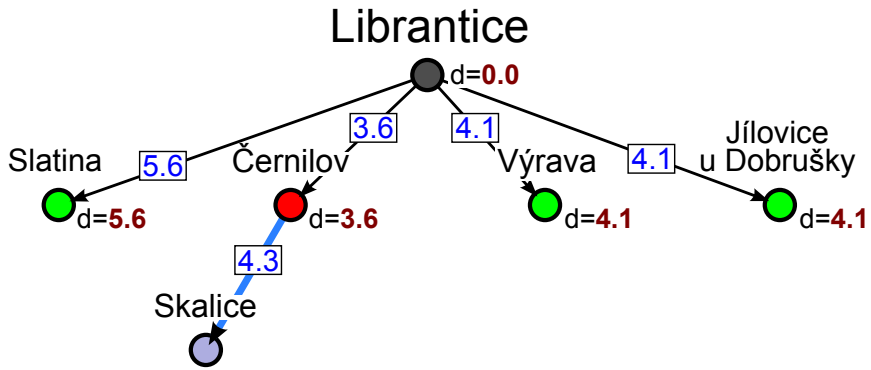




Dijkstra's Algorithm : Example

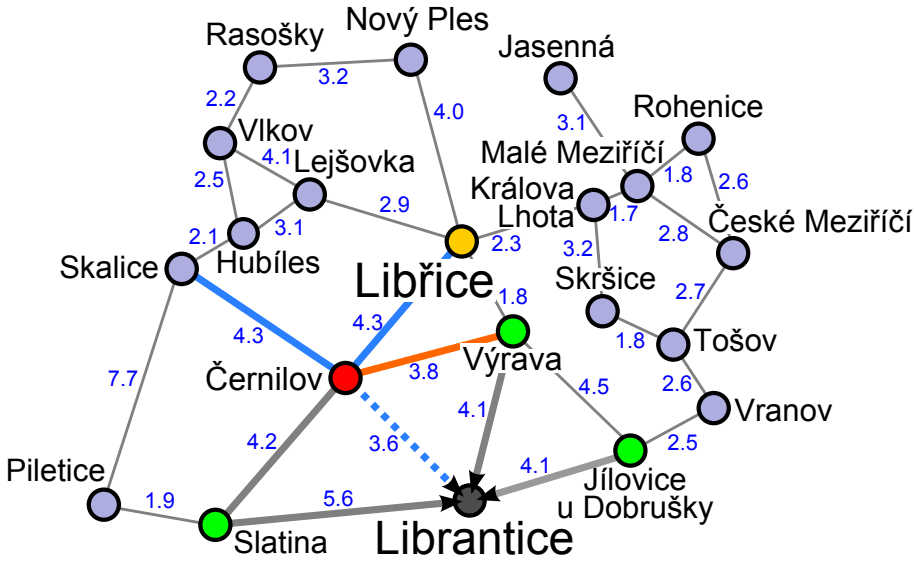
The shortest path from Librantice to Libřice?

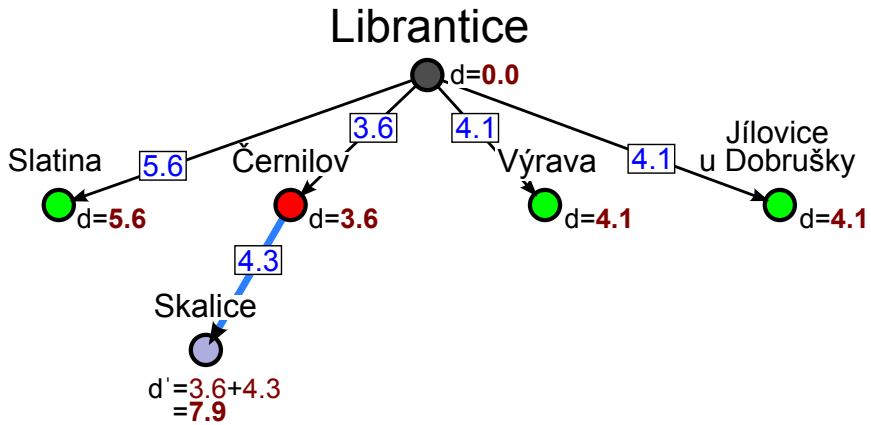




Dijkstra's Algorithm : Example

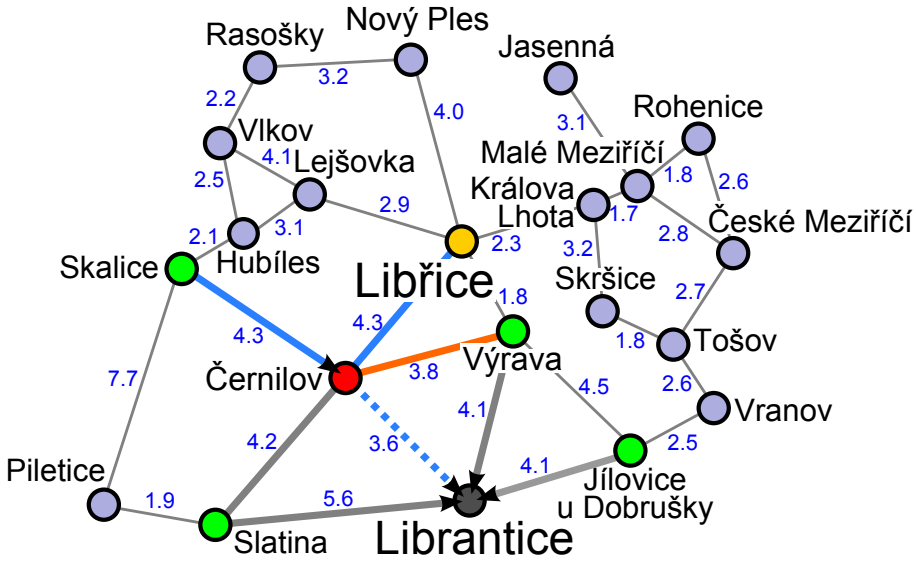
The shortest path from Librantice to Libřice?

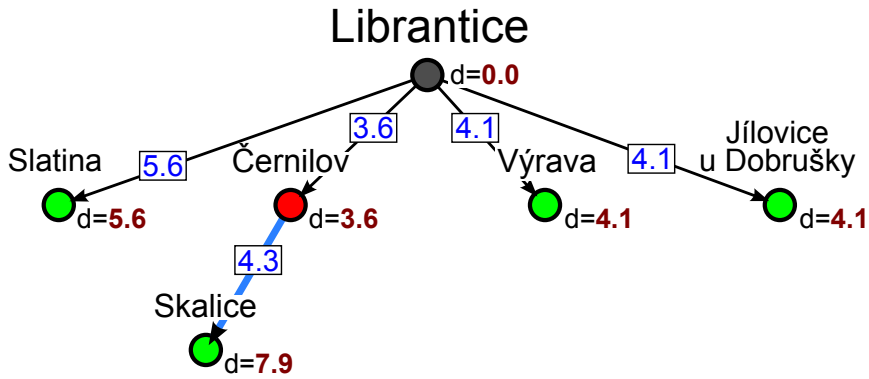




Dijkstra's Algorithm : Example

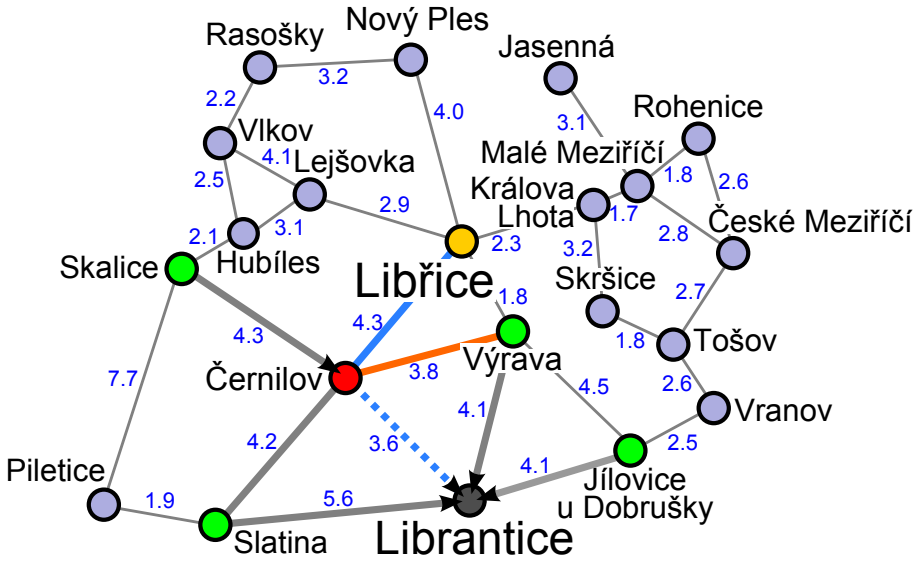
The shortest path from Librantice to Libřice?

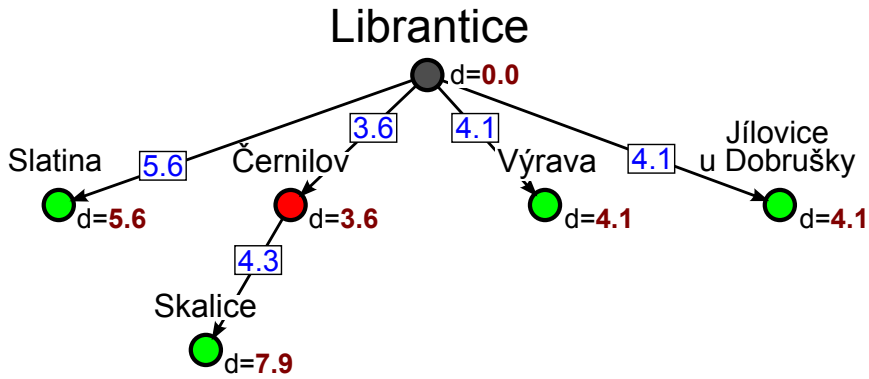




Dijkstra's Algorithm : Example

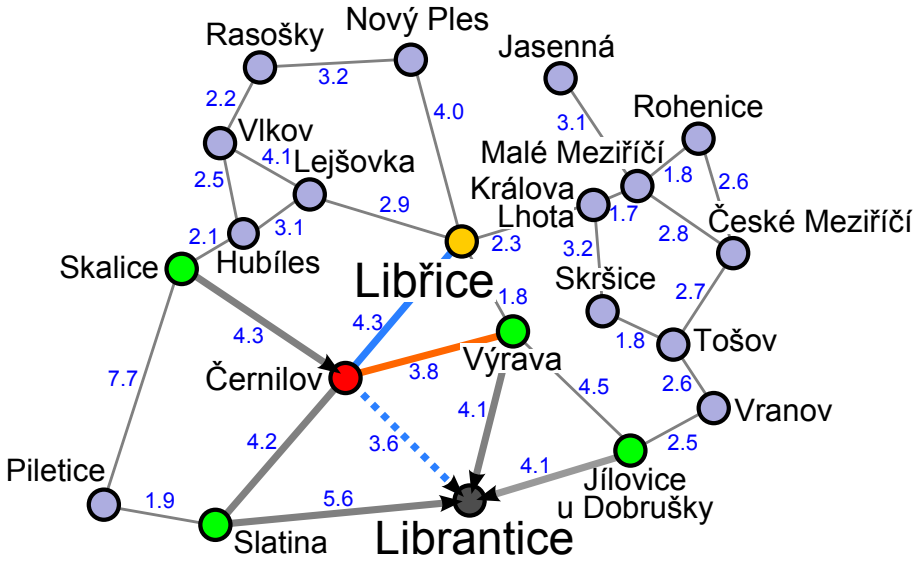
The shortest path from Librantice to Libřice?

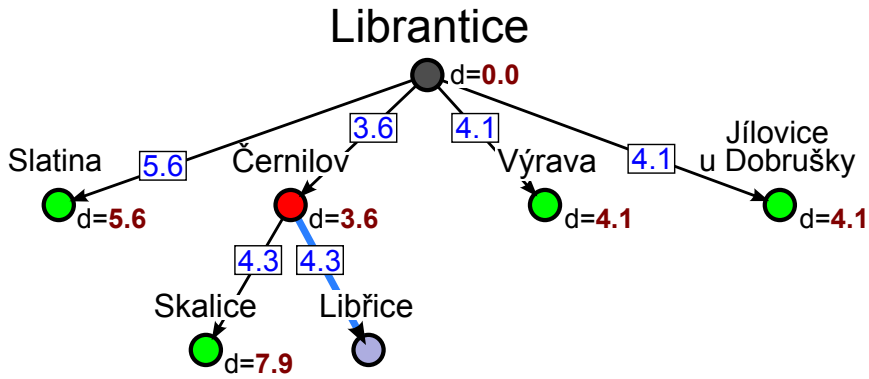




Dijkstra's Algorithm : Example

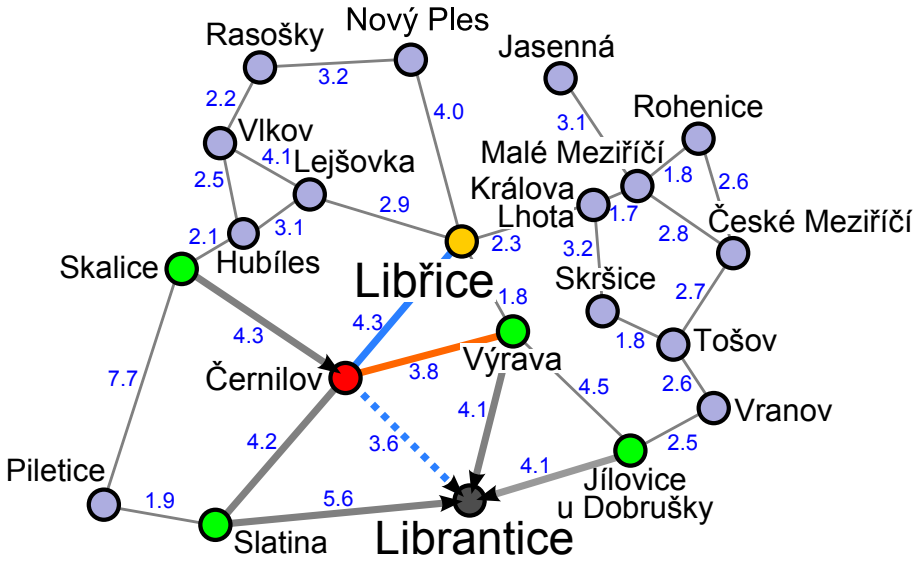
The shortest path from Librantice to Libřice?

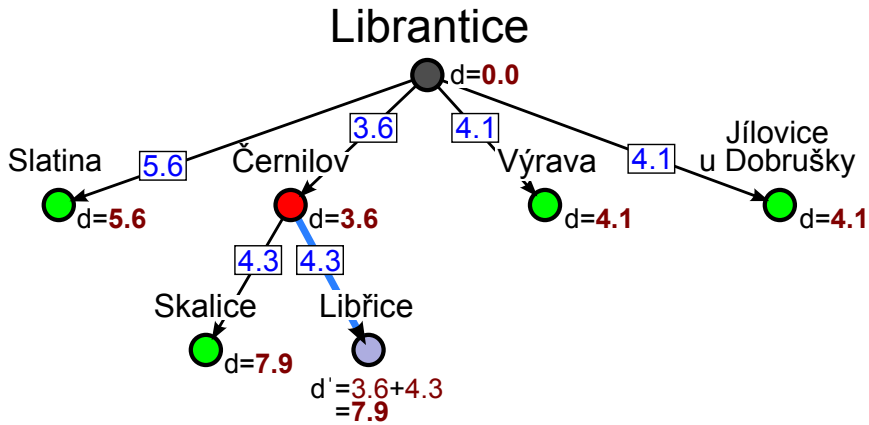




Dijkstra's Algorithm : Example

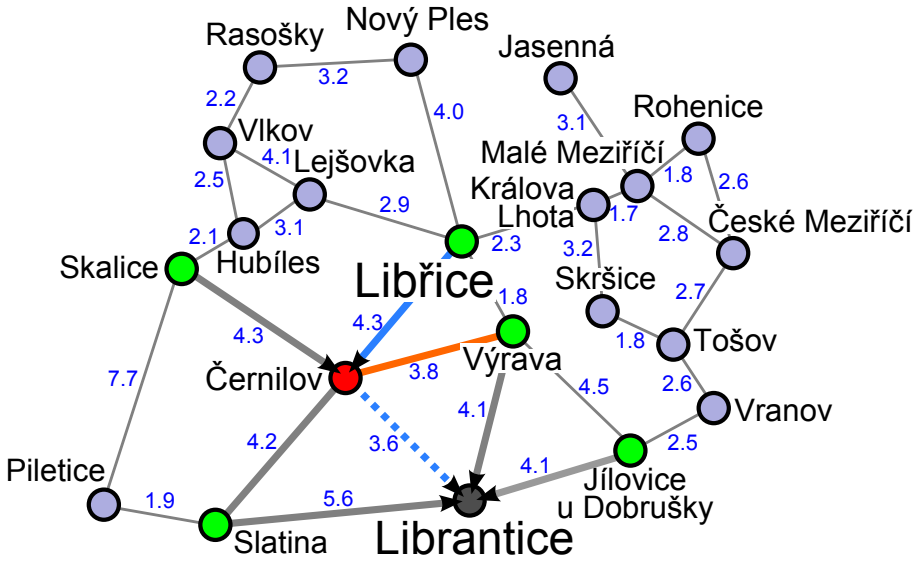
The shortest path from Librantice to Libřice?

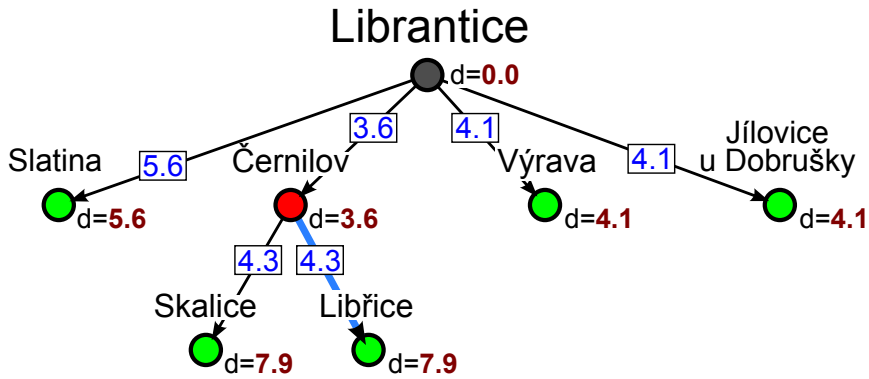




Dijkstra's Algorithm : Example

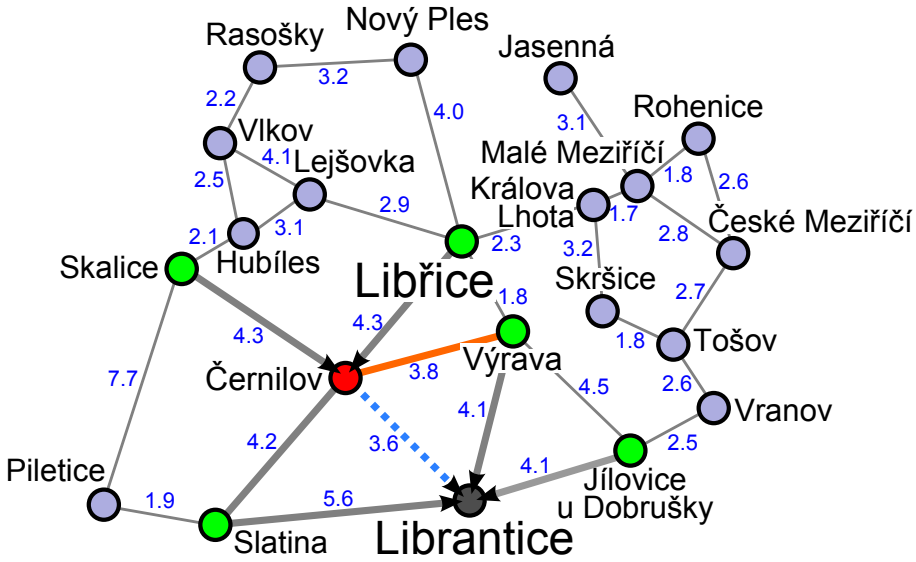
The shortest path from Librantice to Libřice?

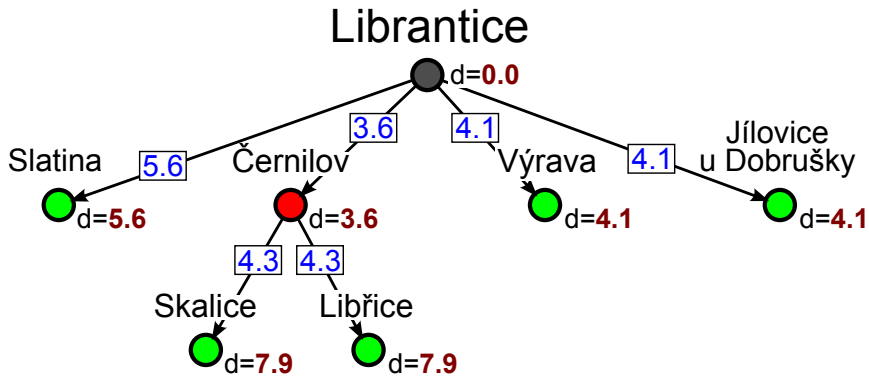




Dijkstra's Algorithm : Example

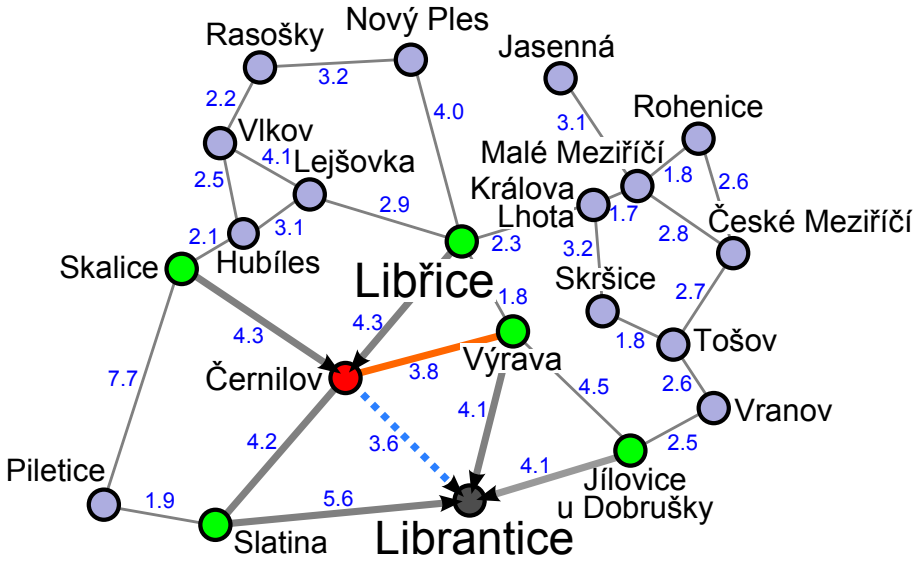
The shortest path from Librantice to Libřice?

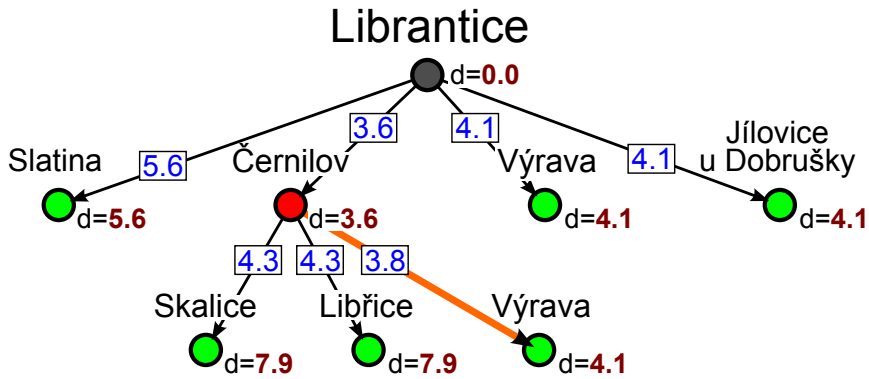




Dijkstra's Algorithm : Example

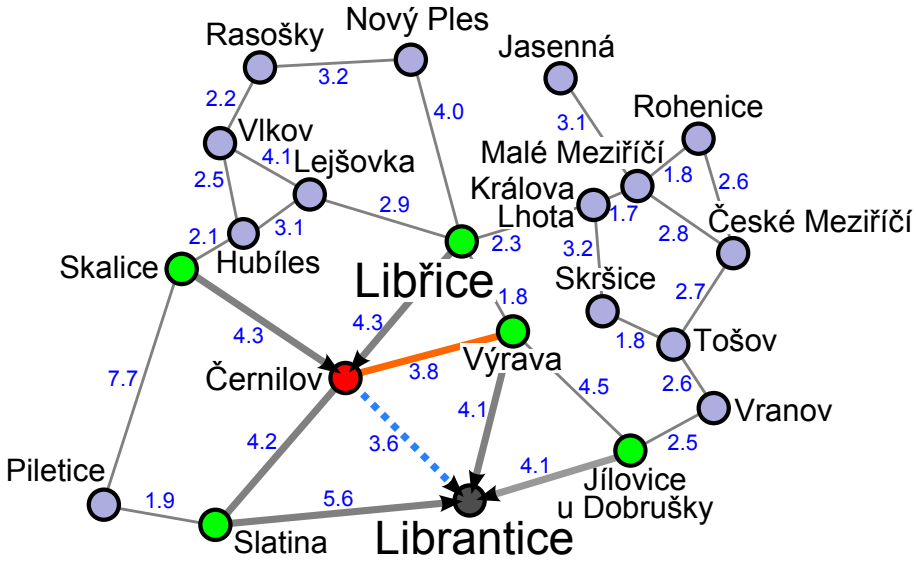
The shortest path from Librantice to Libřice?

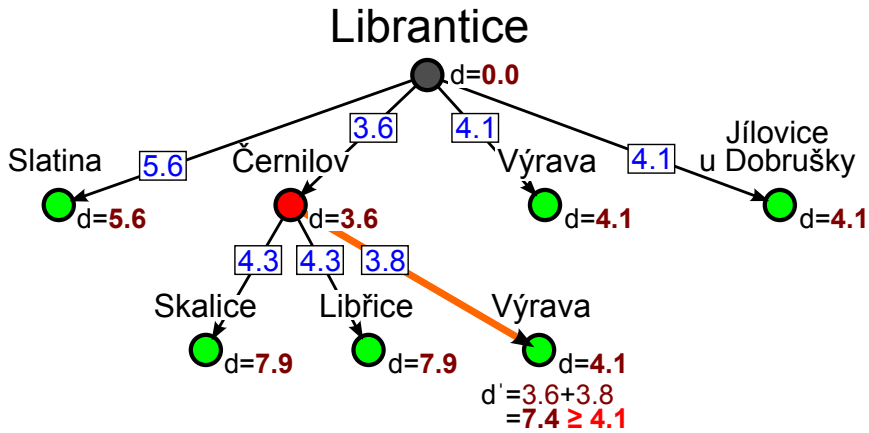




Dijkstra's Algorithm : Example

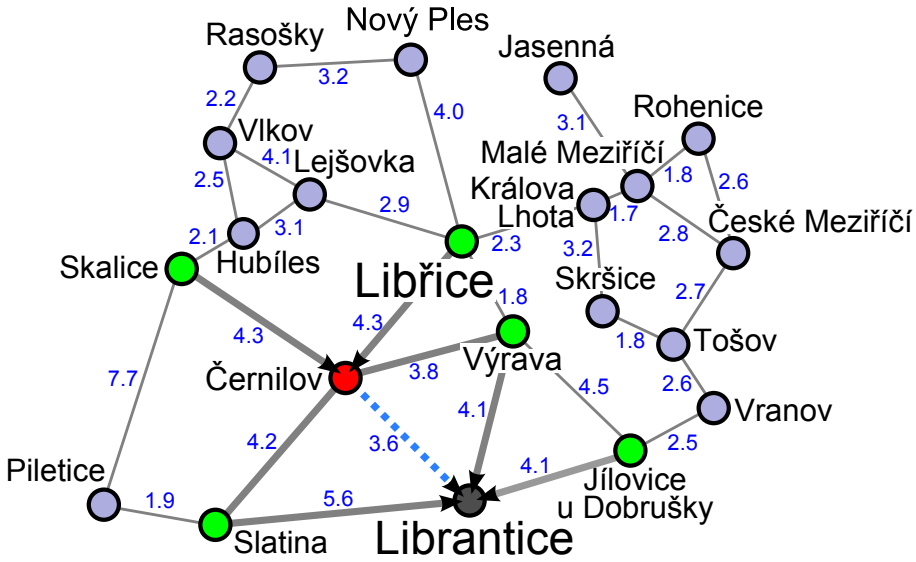
The shortest path from Librantice to Libřice?

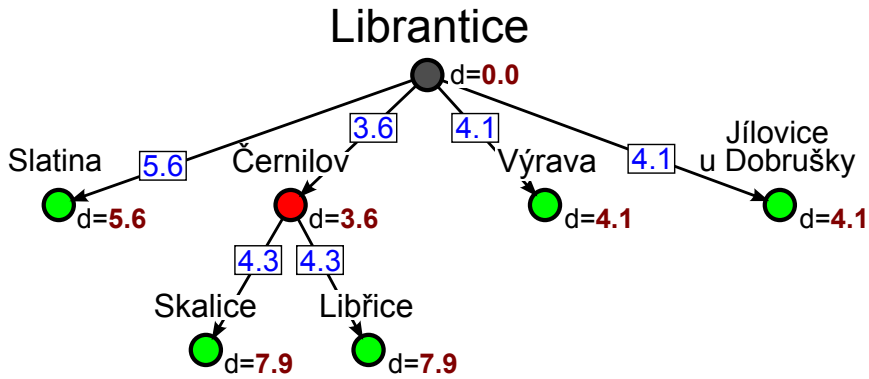




Dijkstra's Algorithm : Example

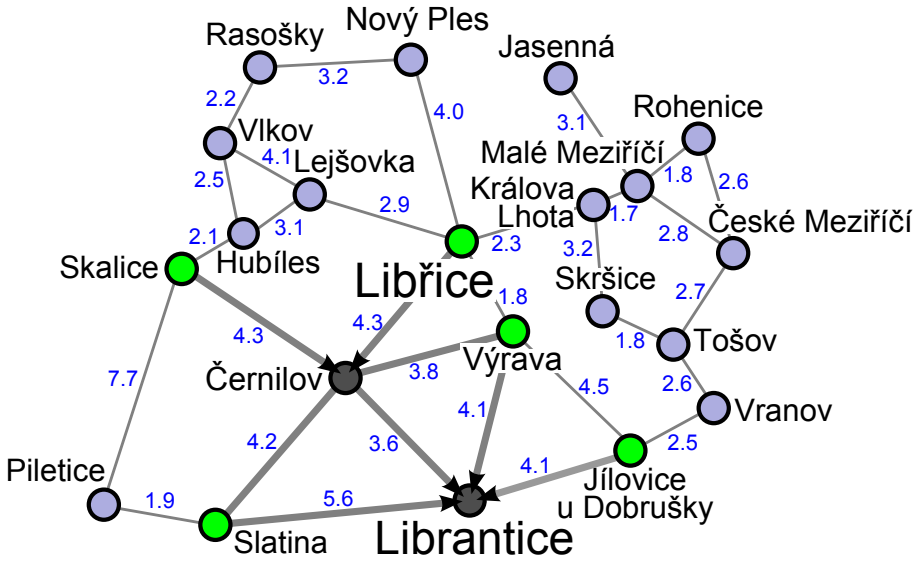
The shortest path from Librantice to Libřice?

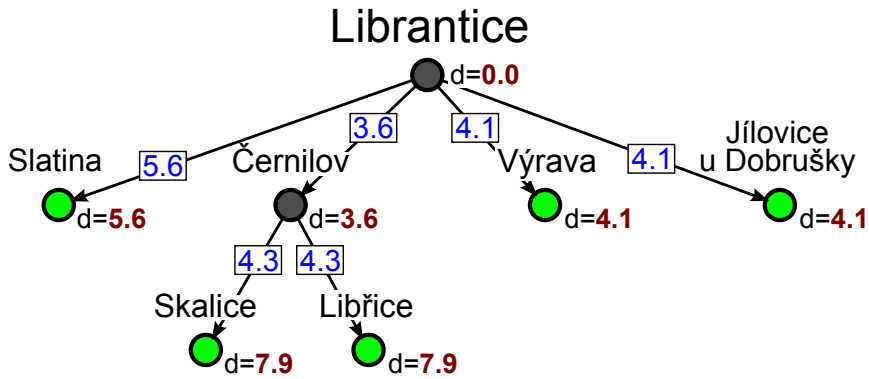




Dijkstra's Algorithm : Example

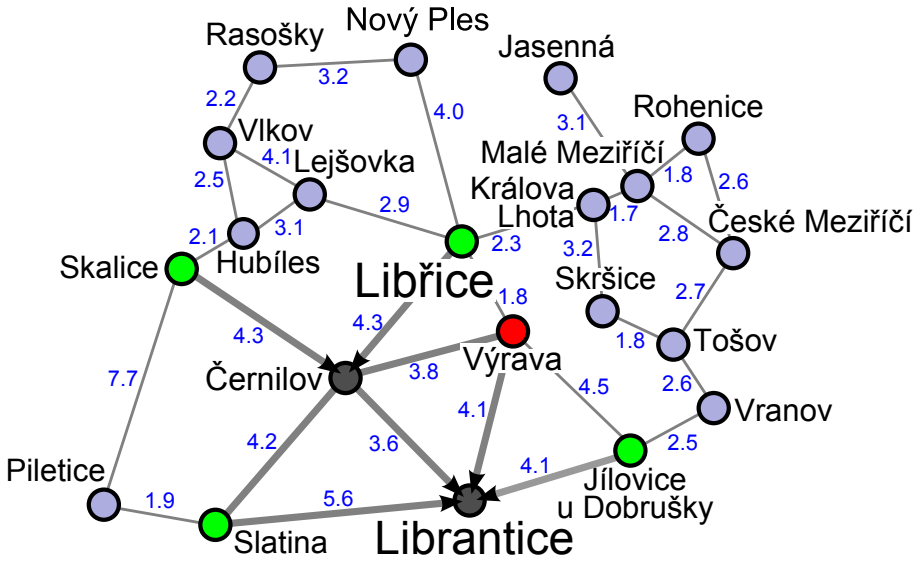
The shortest path from Librantice to Libřice?

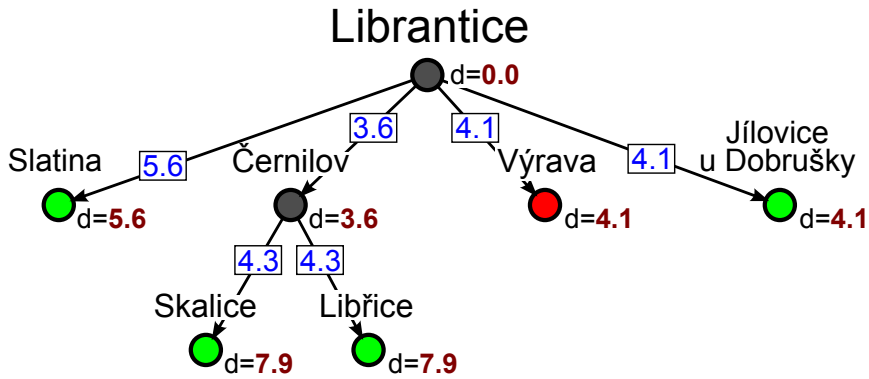




Dijkstra's Algorithm : Example

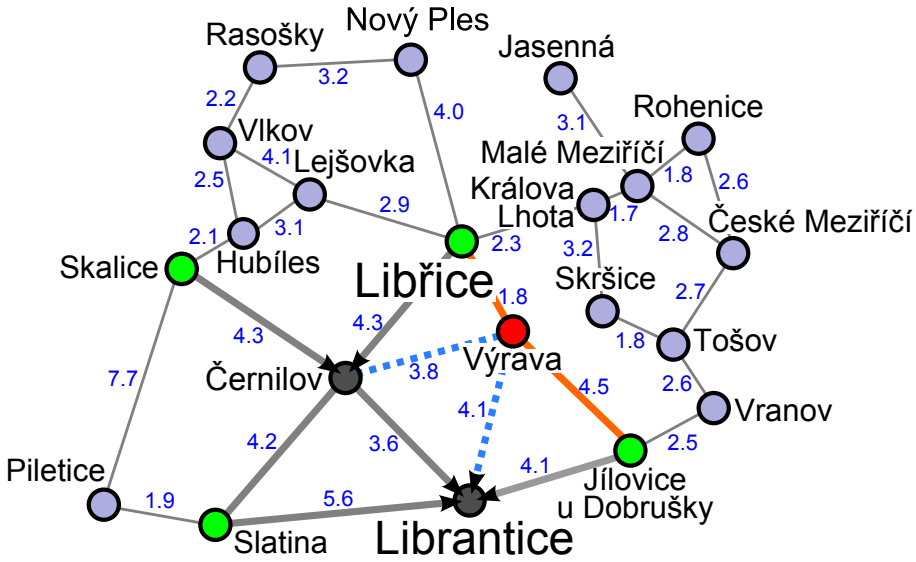
The shortest path from Librantice to Libřice?

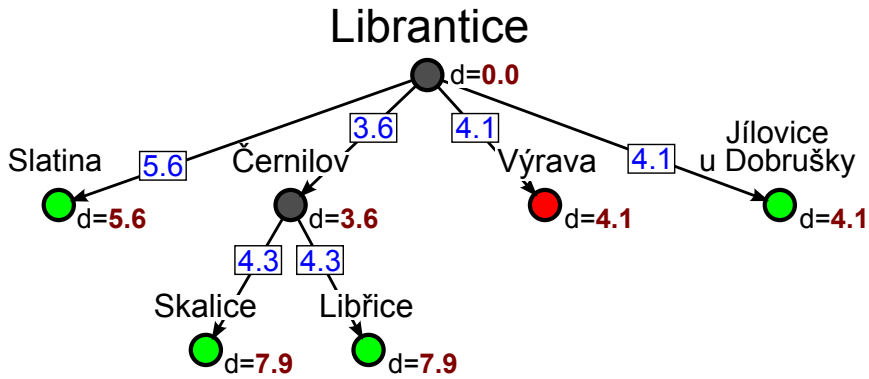




Dijkstra's Algorithm : Example

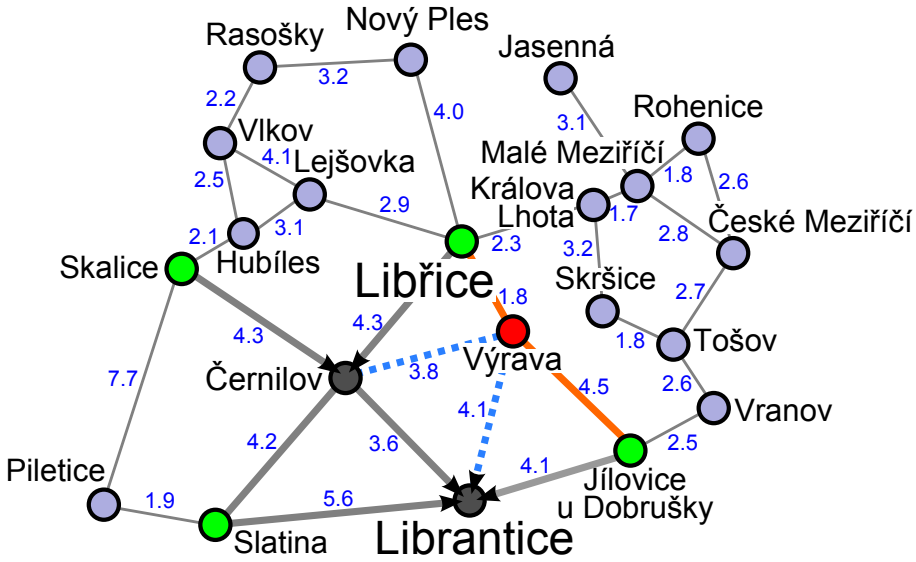
The shortest path from Librantice to Libřice?

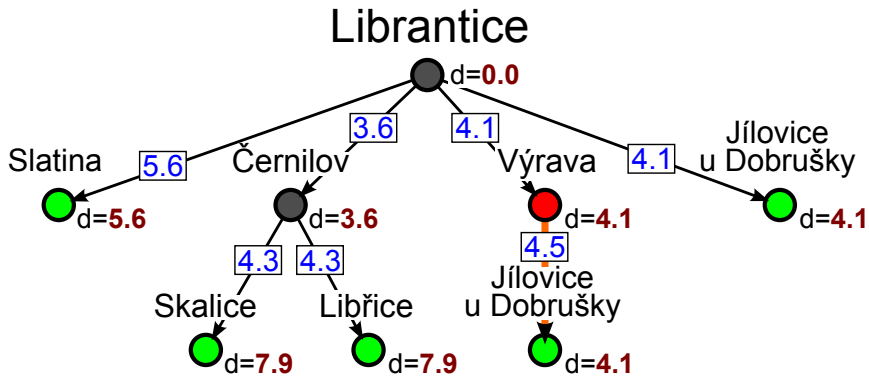




Dijkstra's Algorithm : Example

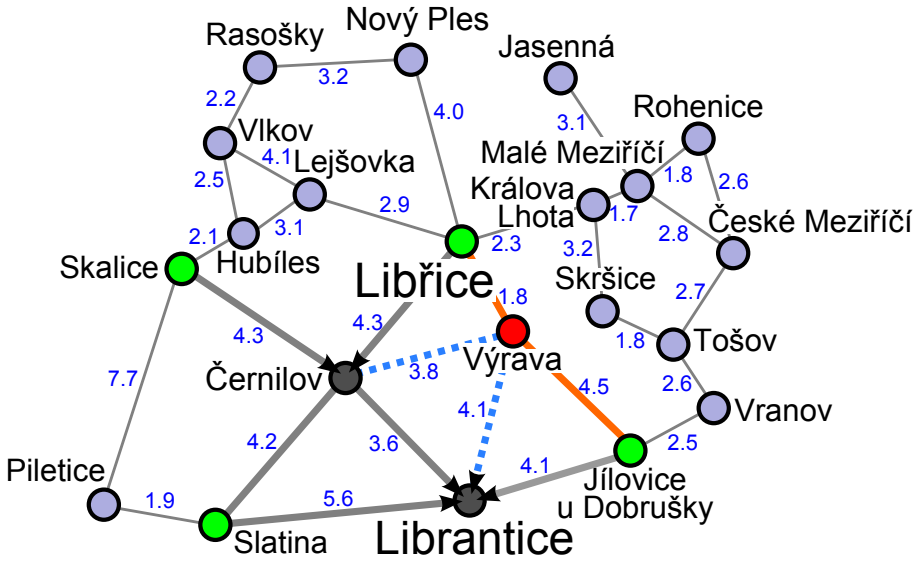
The shortest path from Librantice to Libřice?

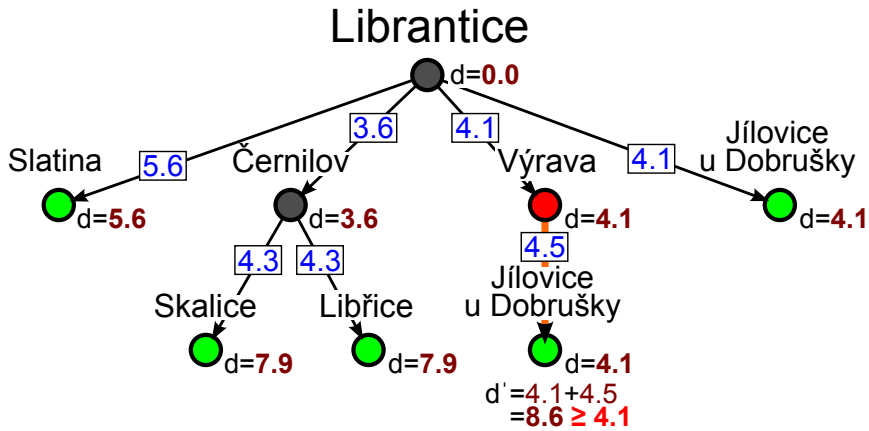




Dijkstra's Algorithm : Example

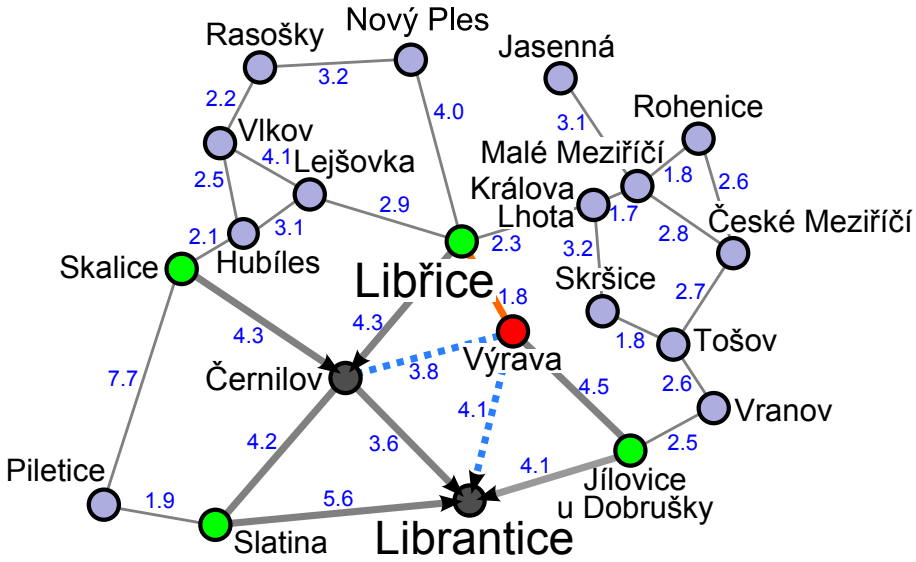
The shortest path from Librantice to Libřice?

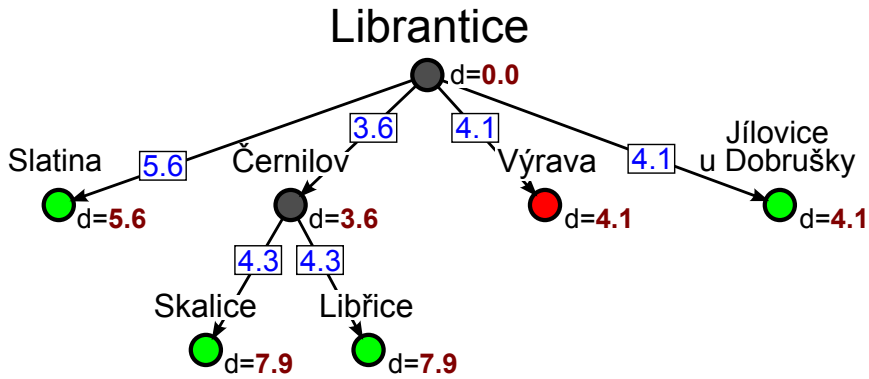




Dijkstra's Algorithm : Example

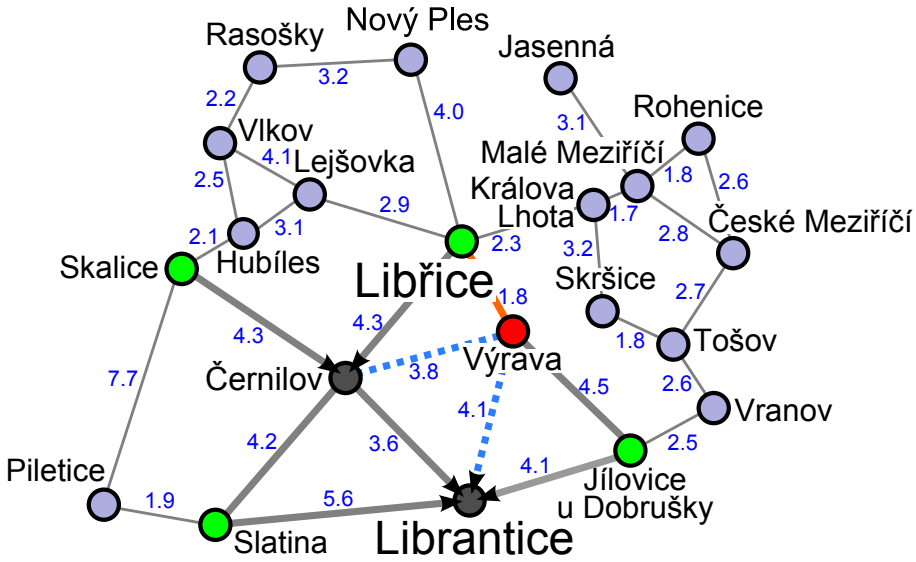
The shortest path from Librantice to Libřice?

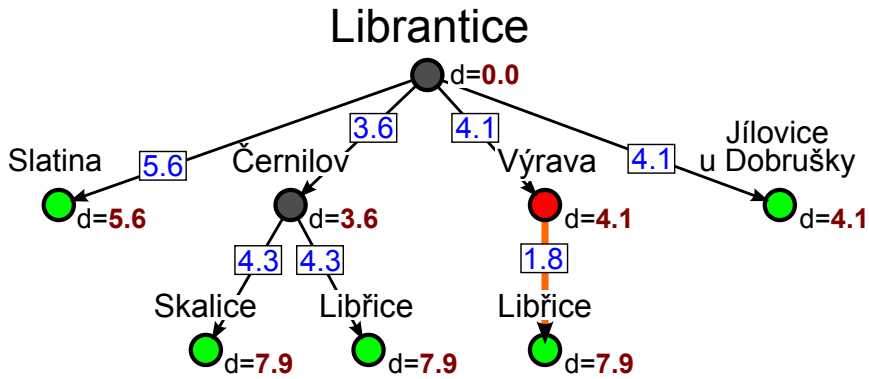




Dijkstra's Algorithm : Example

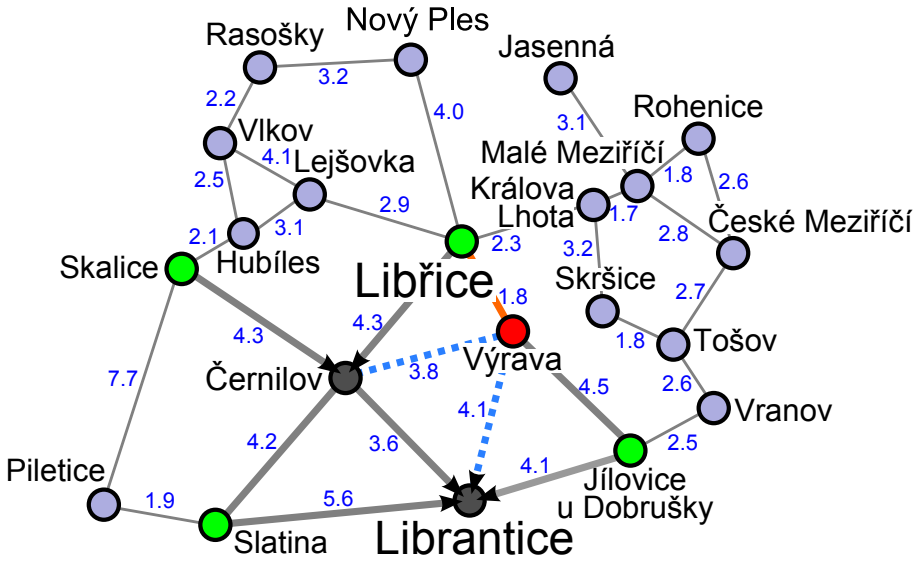
The shortest path from Librantice to Libřice?

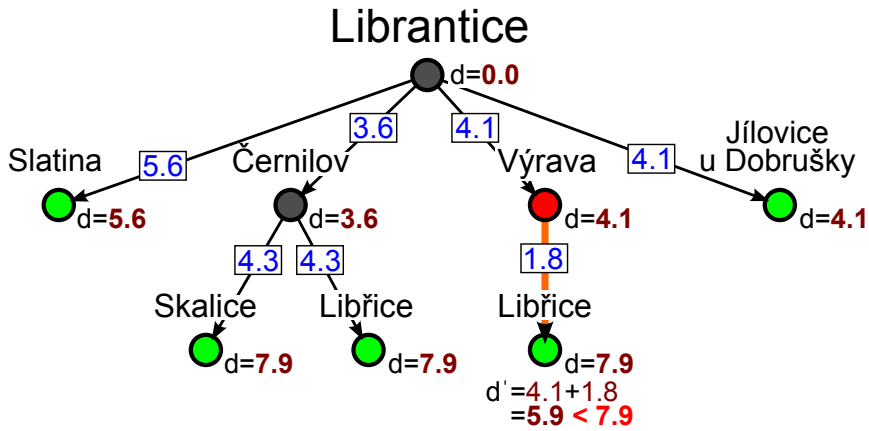




Dijkstra's Algorithm : Example

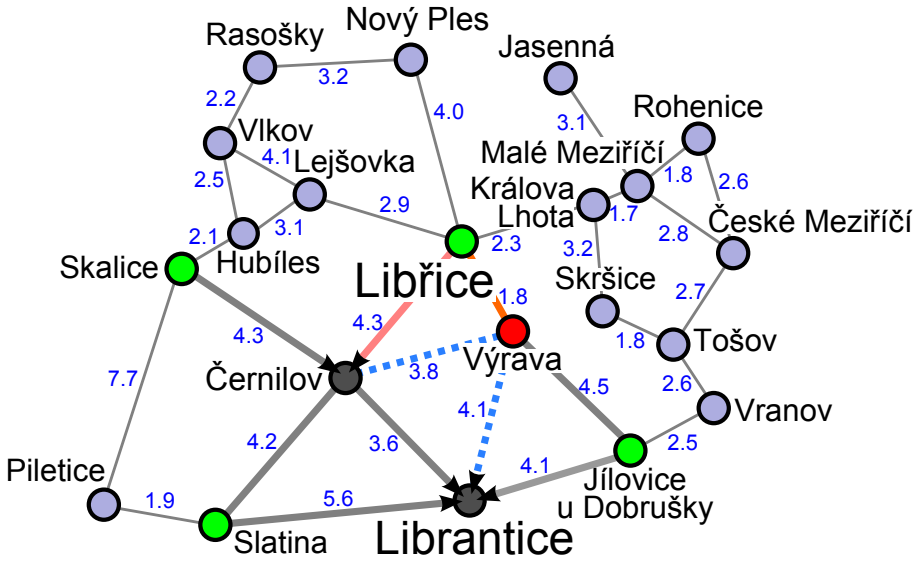
The shortest path from Librantice to Libřice?

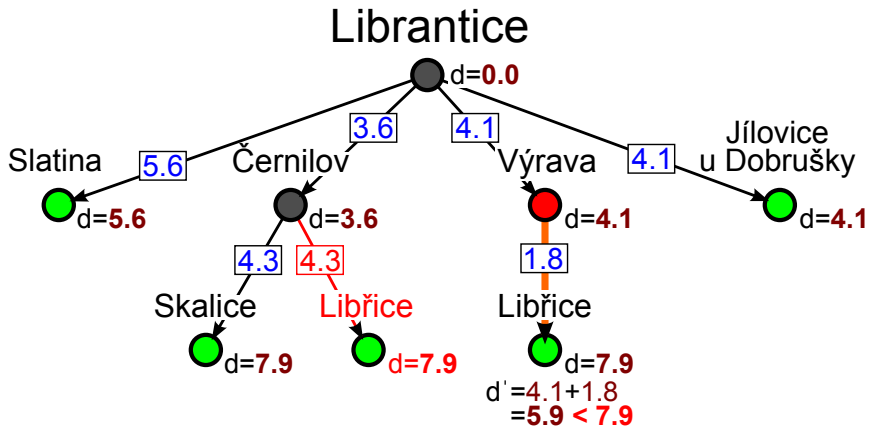




Dijkstra's Algorithm : Example

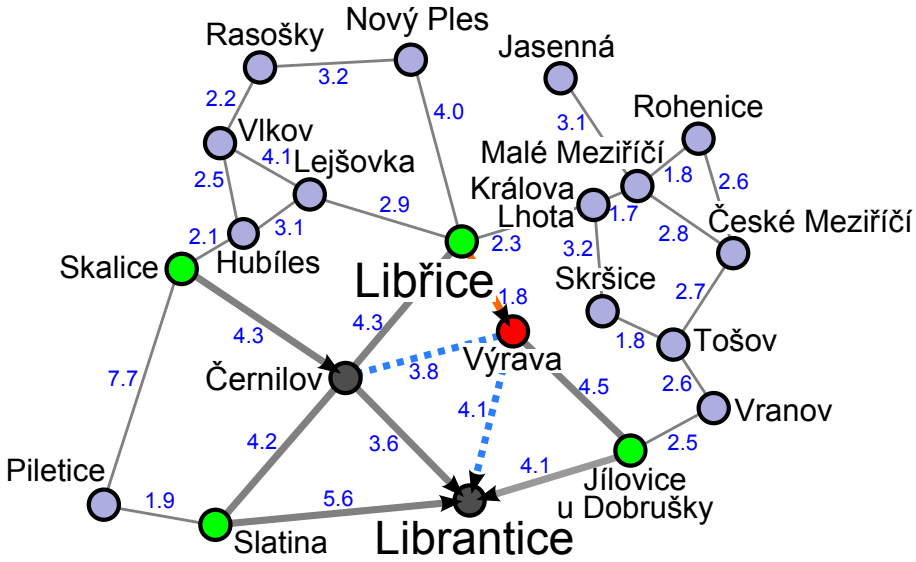
The shortest path from Librantice to Libřice?

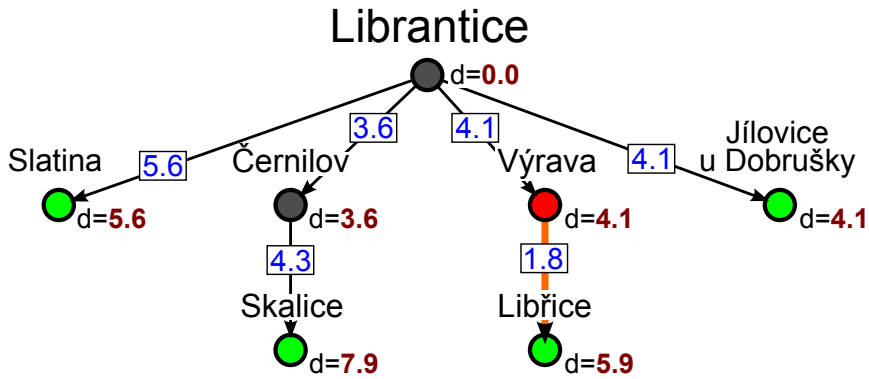




Dijkstra's Algorithm : Example

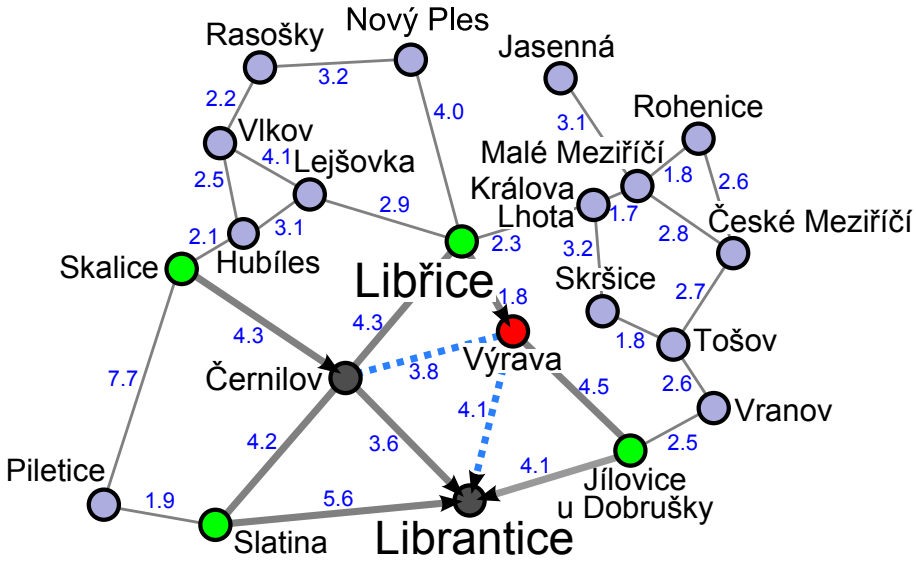
The shortest path from Librantice to Libřice?

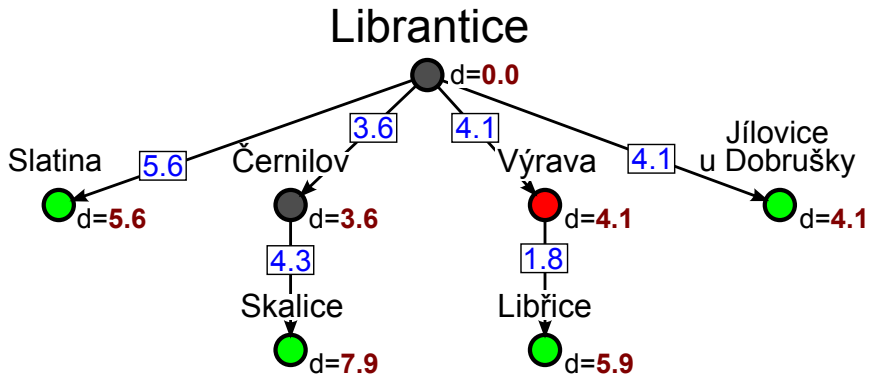




Dijkstra's Algorithm : Example

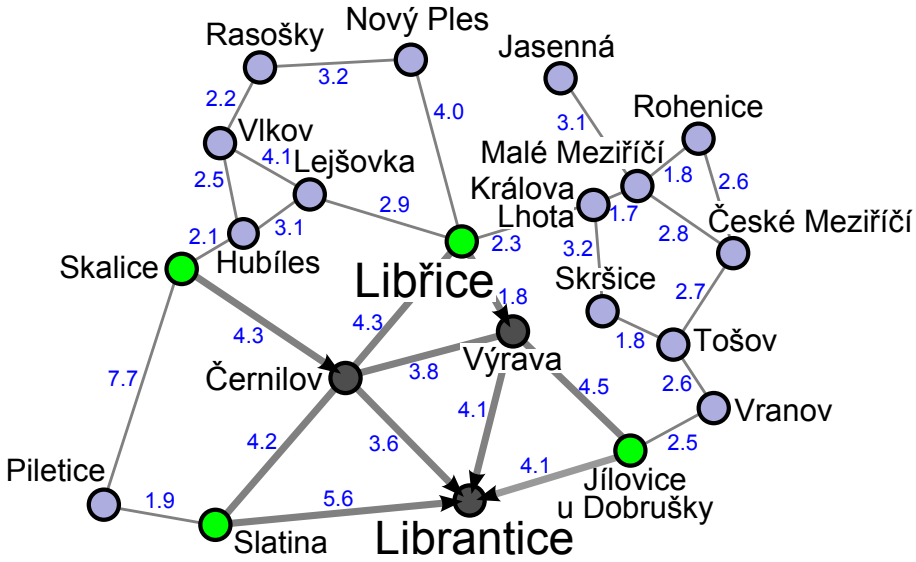
The shortest path from Librantice to Libřice?

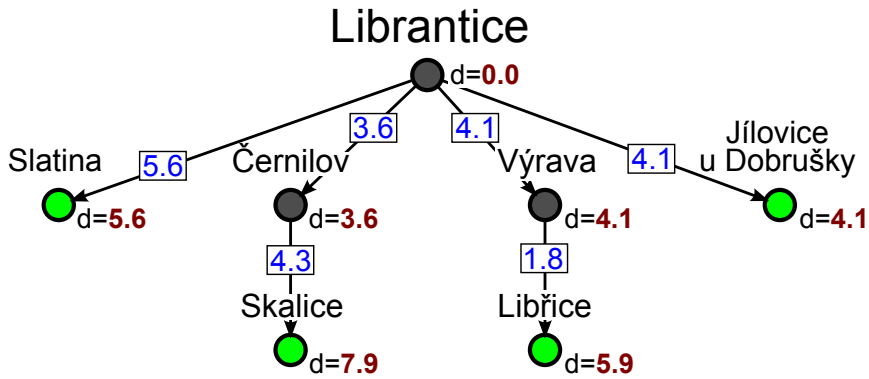




Dijkstra's Algorithm : Example

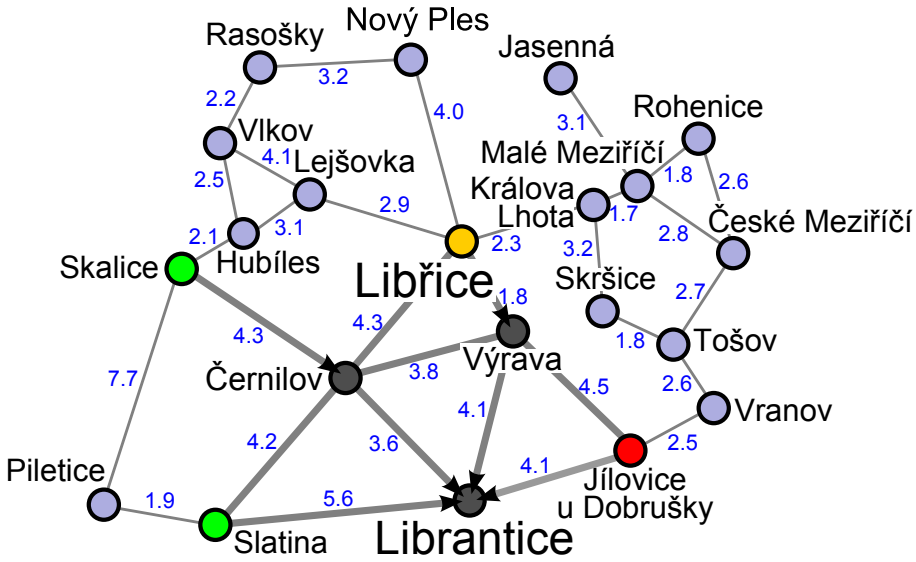
The shortest path from Librantice to Libřice?

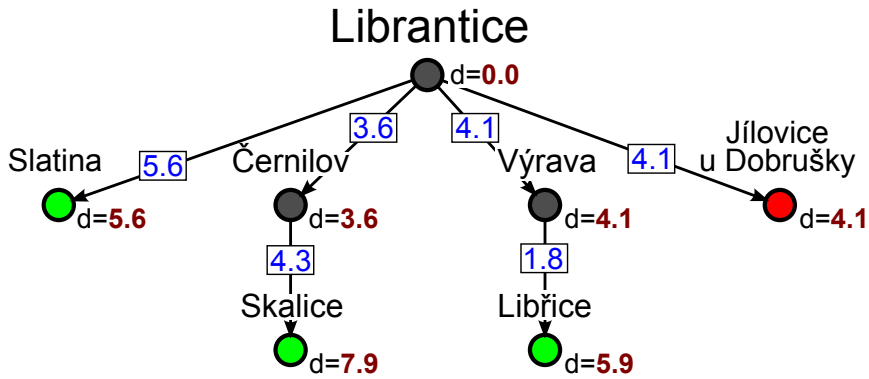




Dijkstra's Algorithm : Example

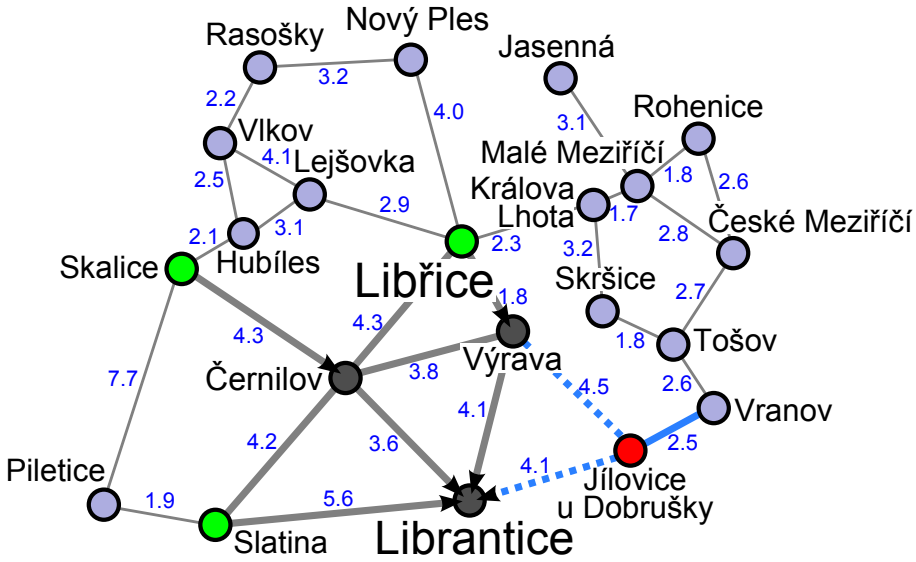
The shortest path from Librantice to Libřice?

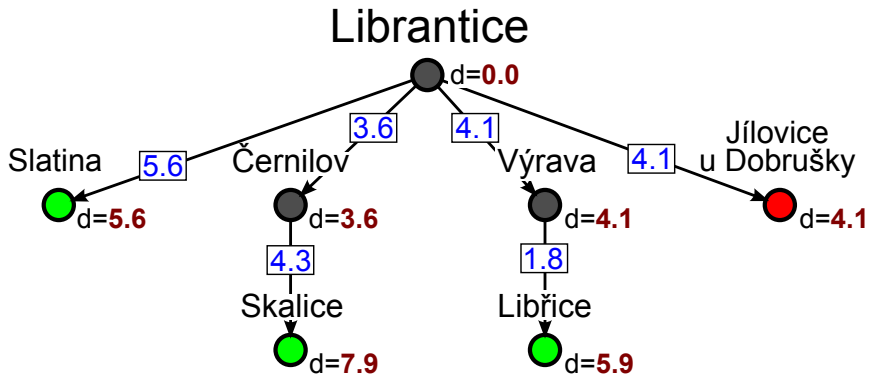




Dijkstra's Algorithm : Example

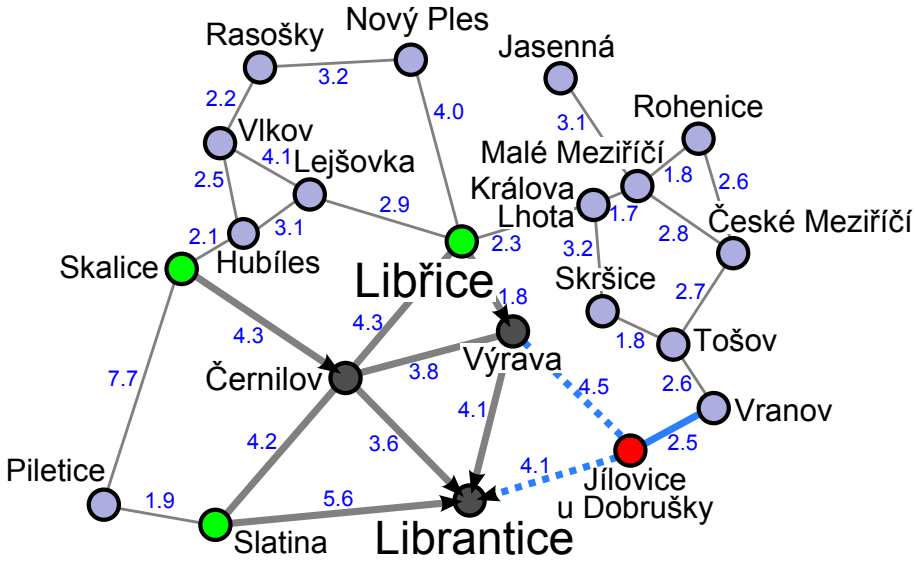
The shortest path from Librantice to Libřice?

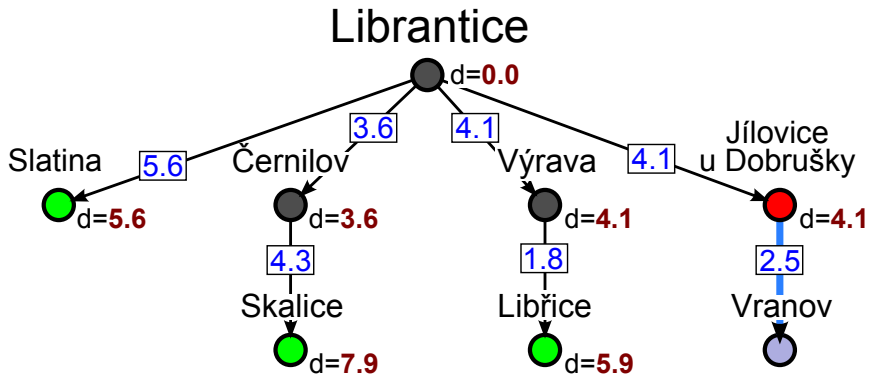




Dijkstra's Algorithm : Example

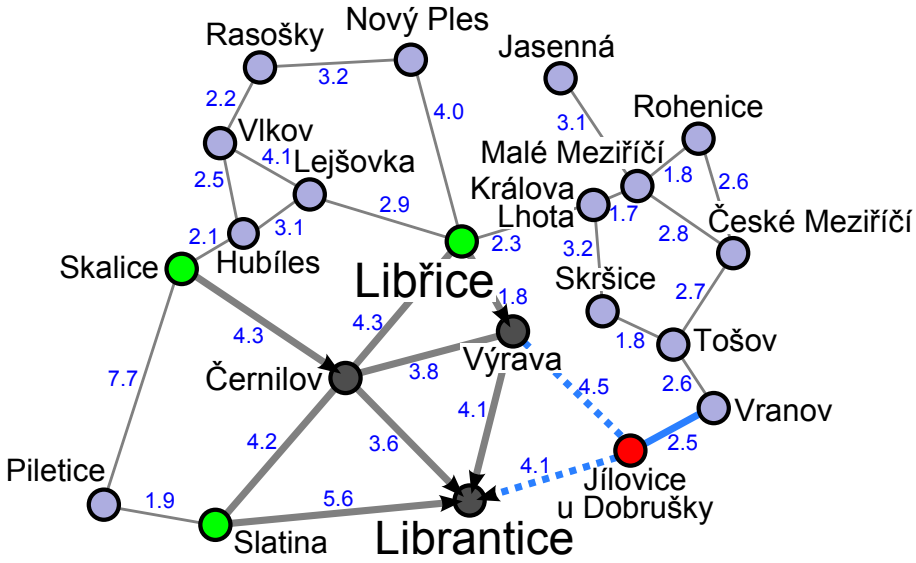
The shortest path from Librantice to Libřice?

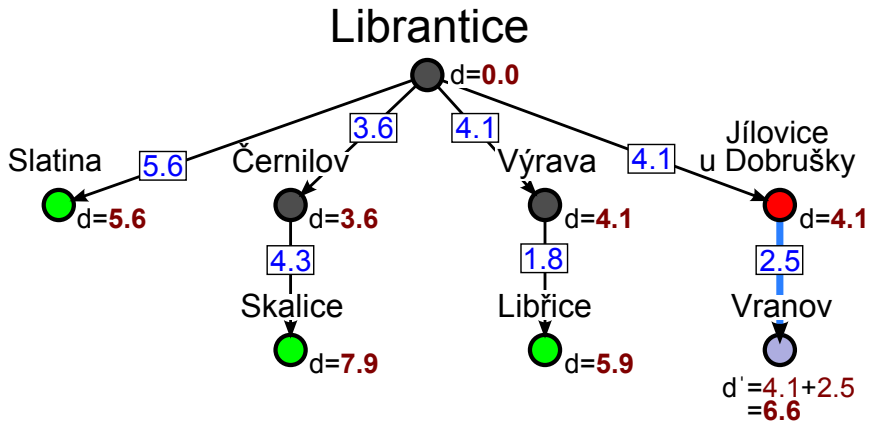




Dijkstra's Algorithm : Example

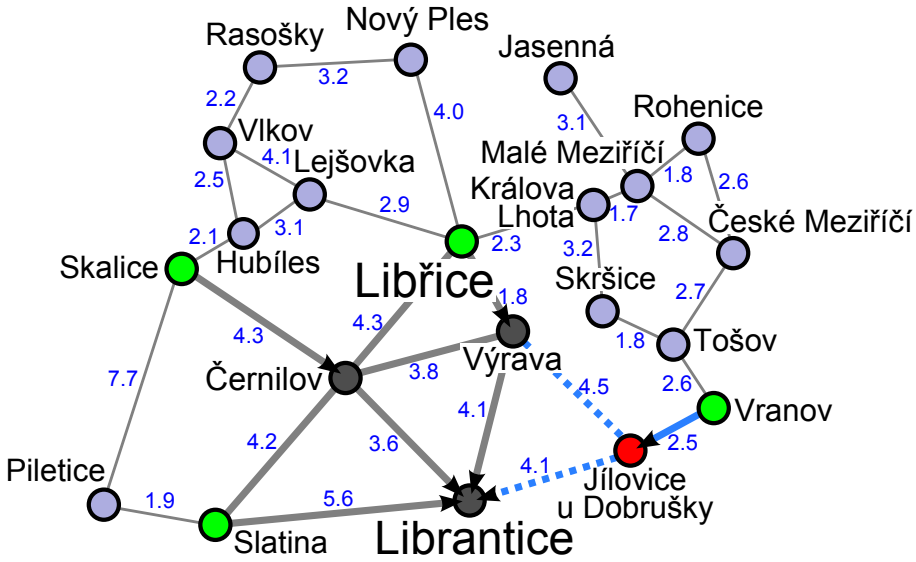
The shortest path from Librantice to Libřice?

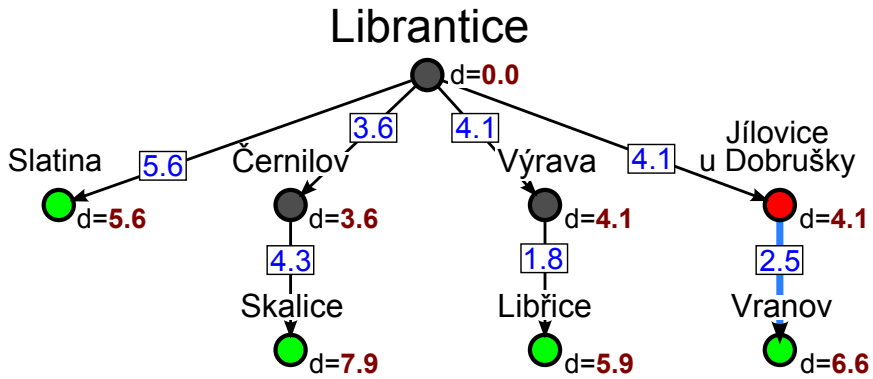




Dijkstra's Algorithm : Example

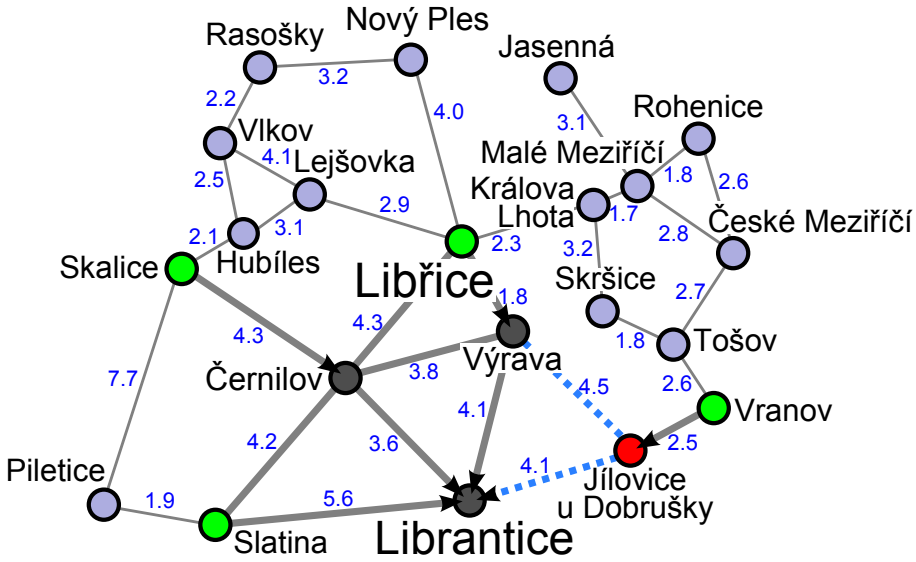
The shortest path from Librantice to Libřice?

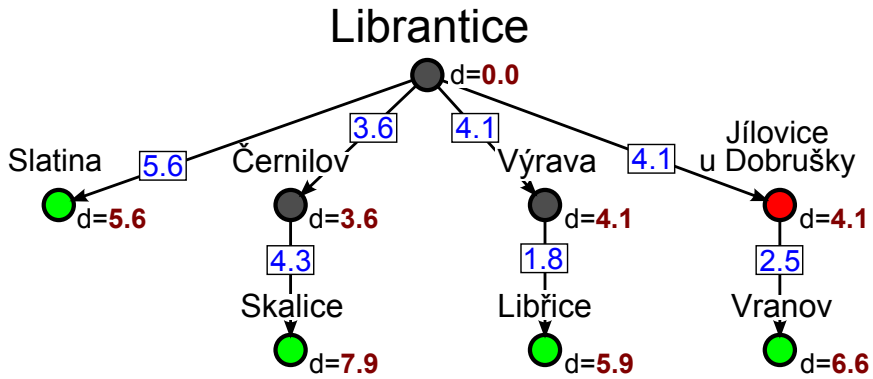




Dijkstra's Algorithm : Example

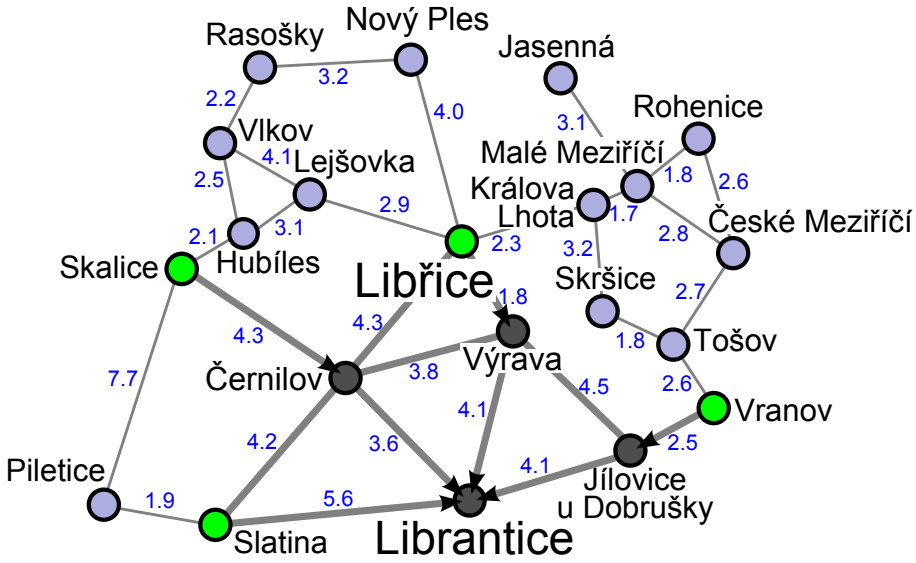
The shortest path from Librantice to Libřice?

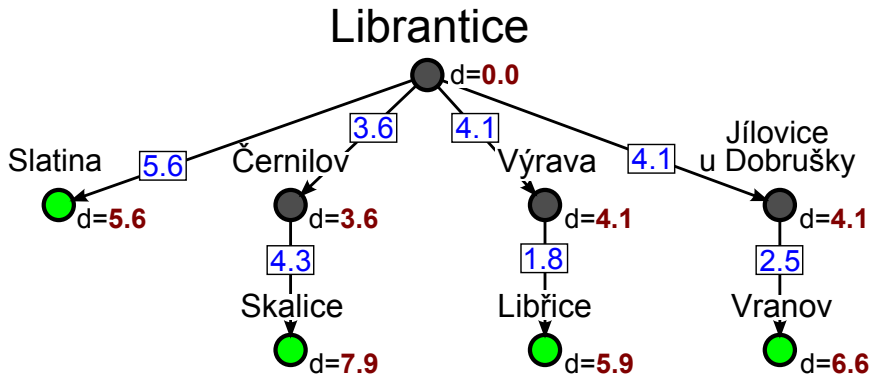




Dijkstra's Algorithm : Example

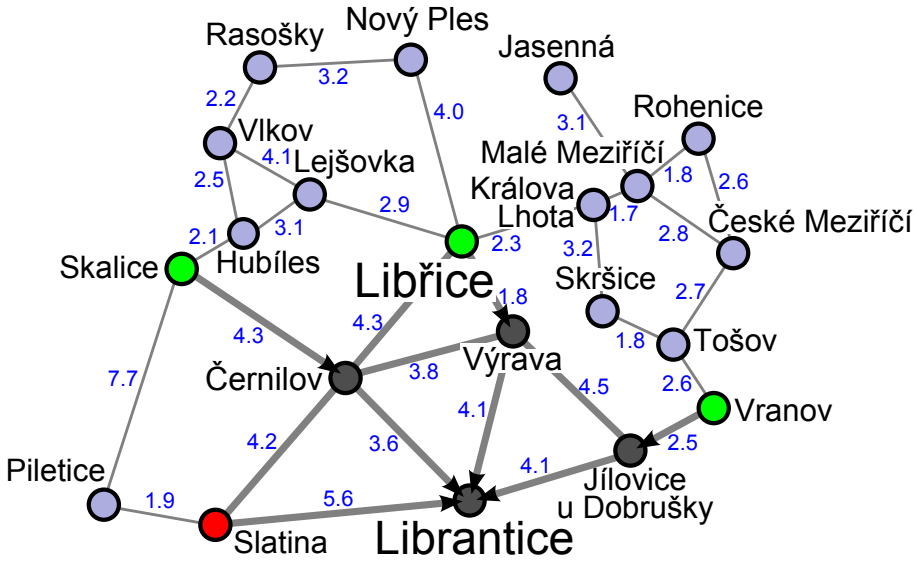
The shortest path from Librantice to Libřice?

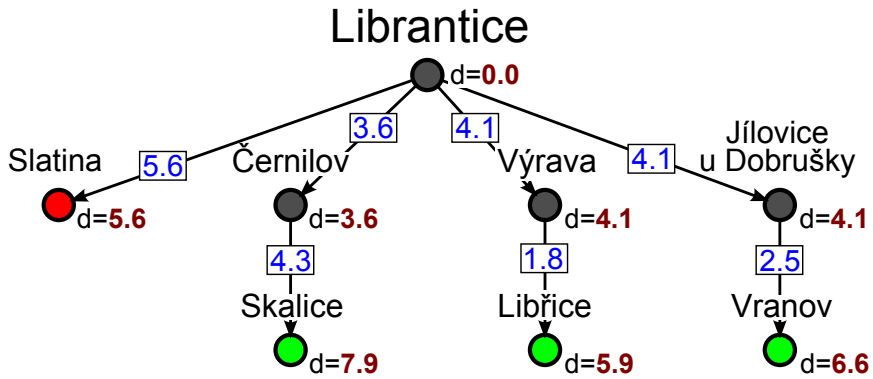




Dijkstra's Algorithm : Example

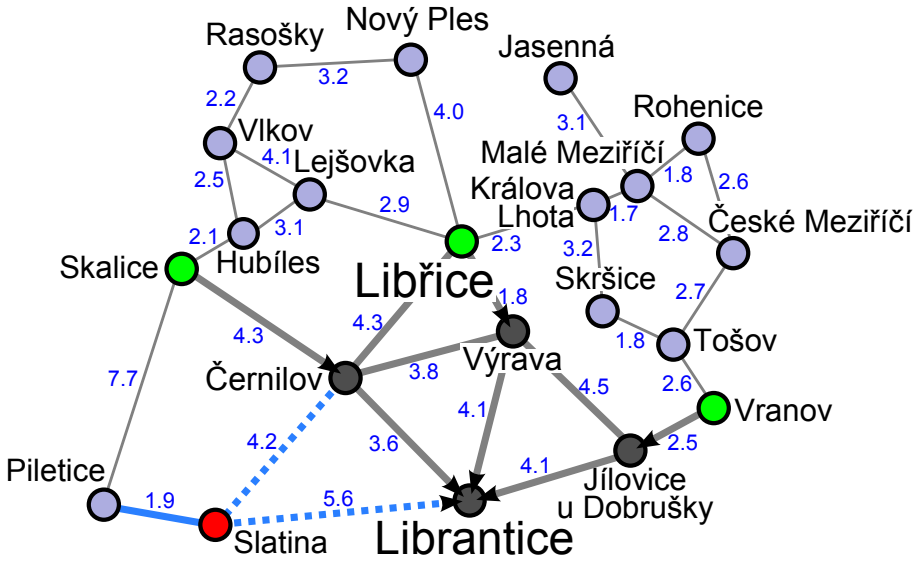
The shortest path from Librantice to Libřice?

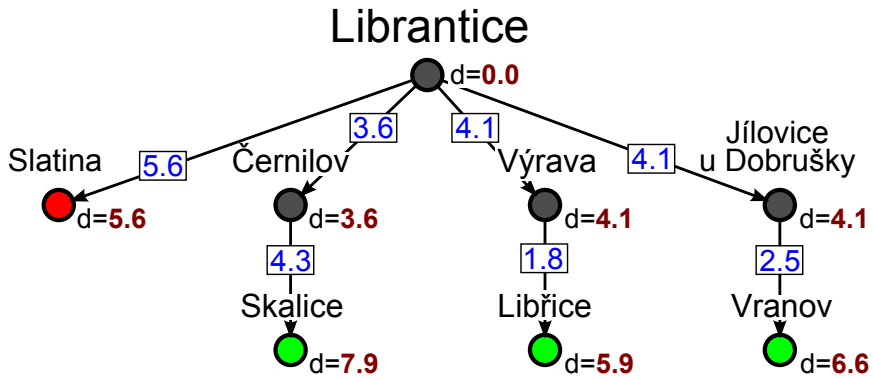




Dijkstra's Algorithm : Example

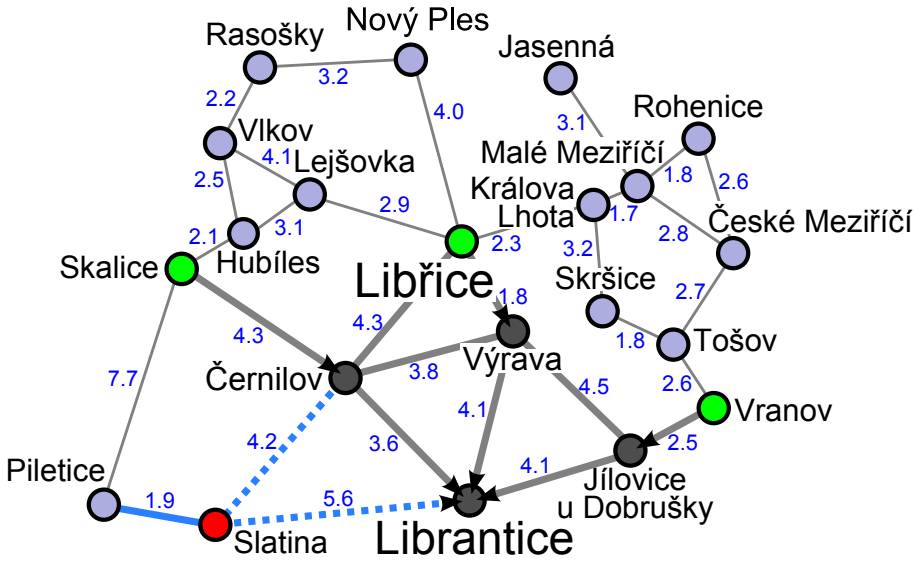
The shortest path from Librantice to Libřice?

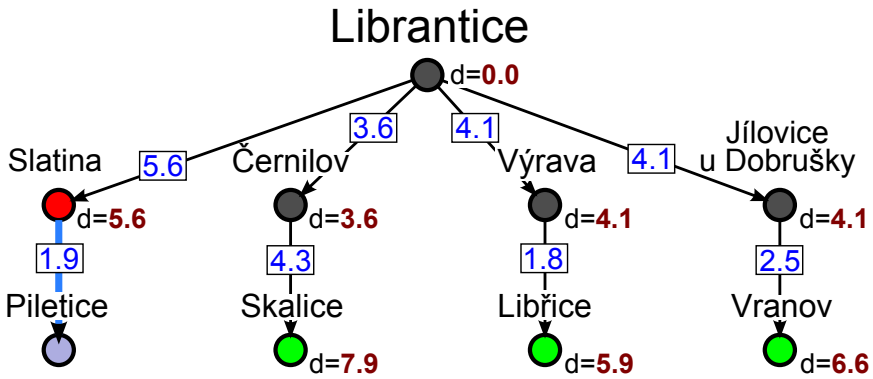




Dijkstra's Algorithm : Example

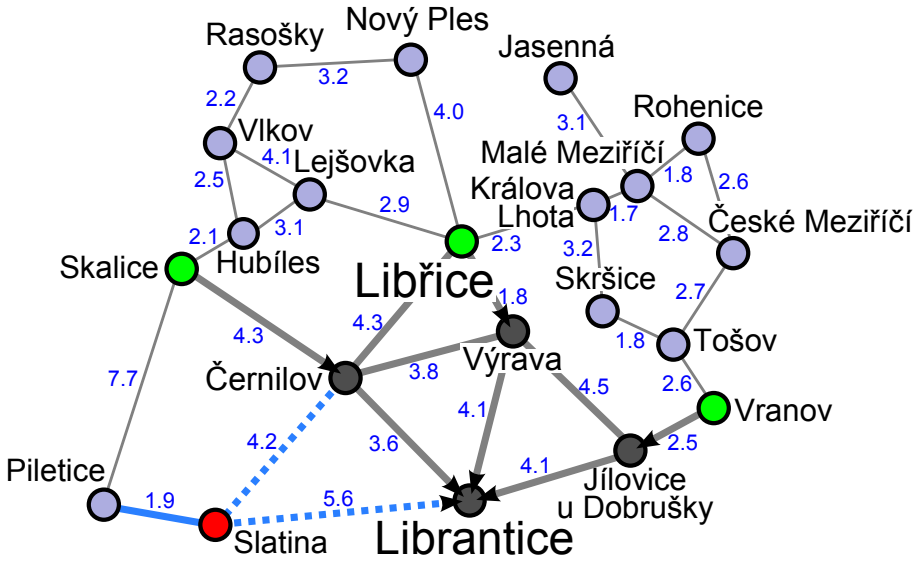
The shortest path from Librantice to Libřice?

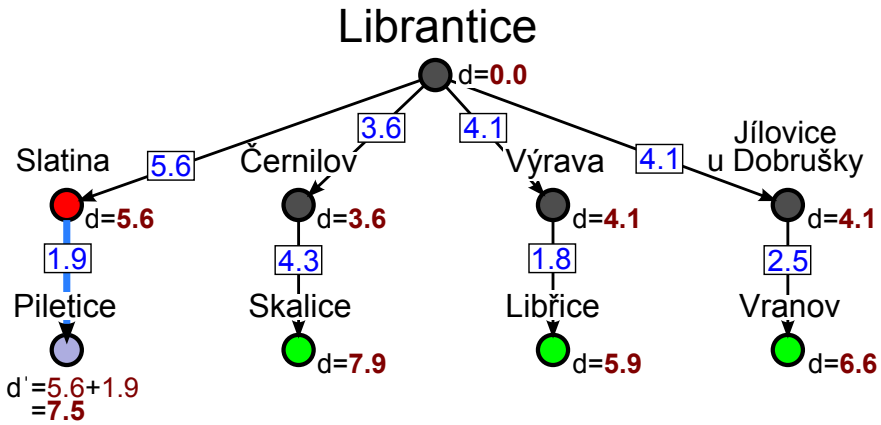




Dijkstra's Algorithm : Example

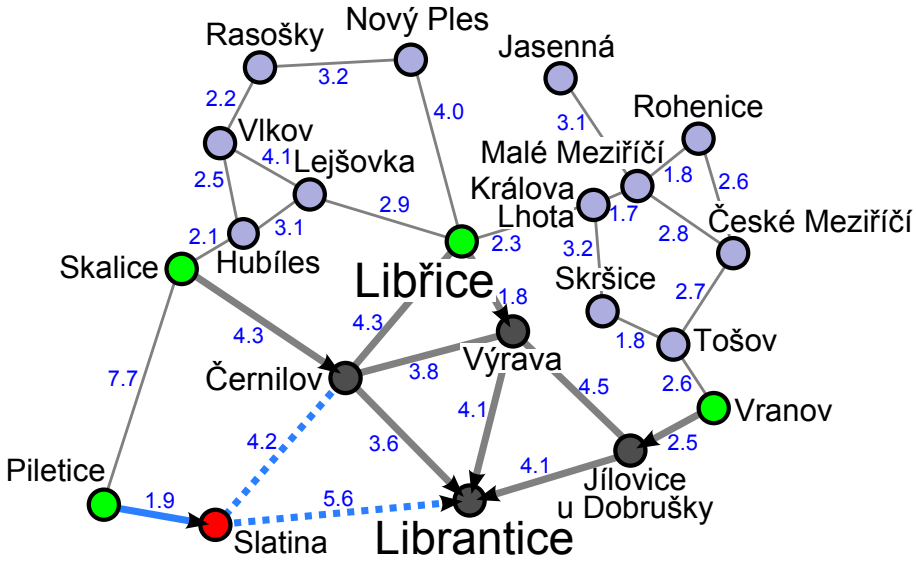
The shortest path from Librantice to Libřice?

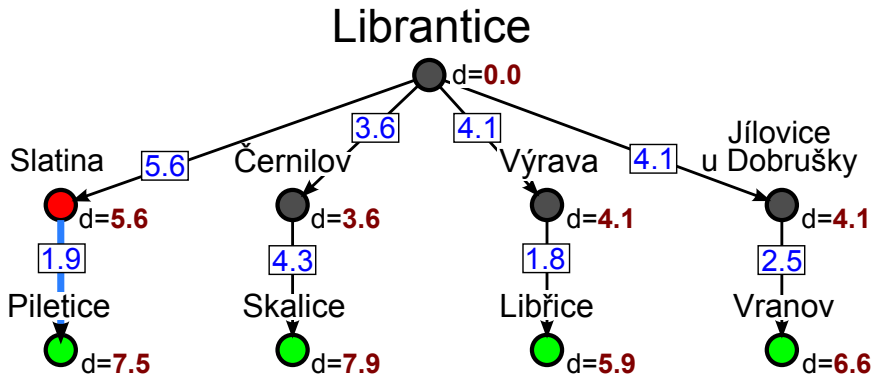




Dijkstra's Algorithm : Example

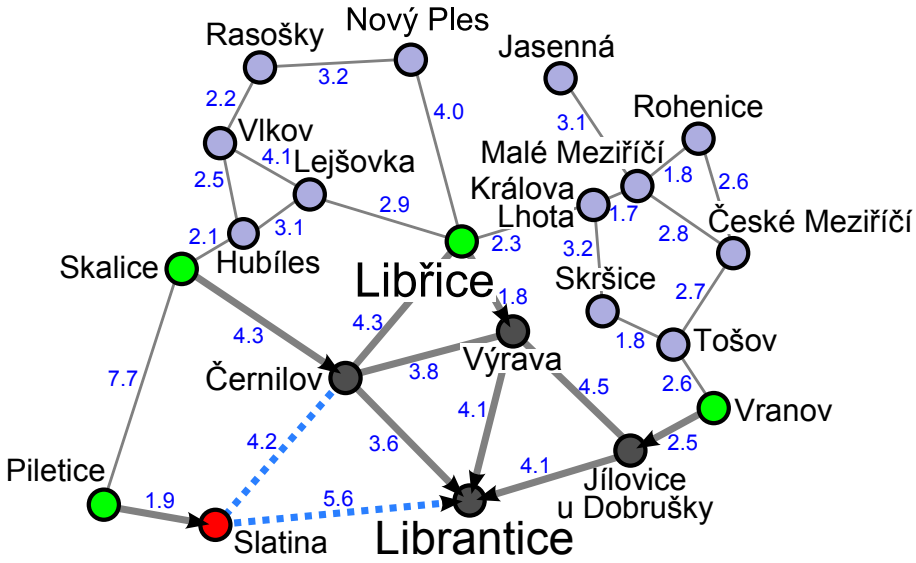
The shortest path from Librantice to Libřice?

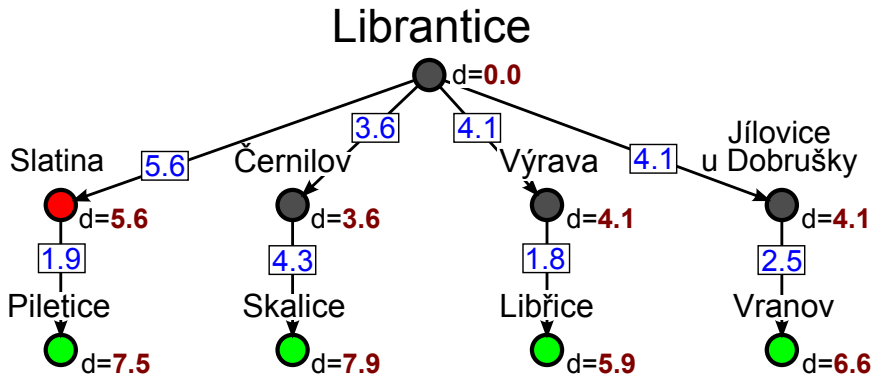




Dijkstra's Algorithm : Example

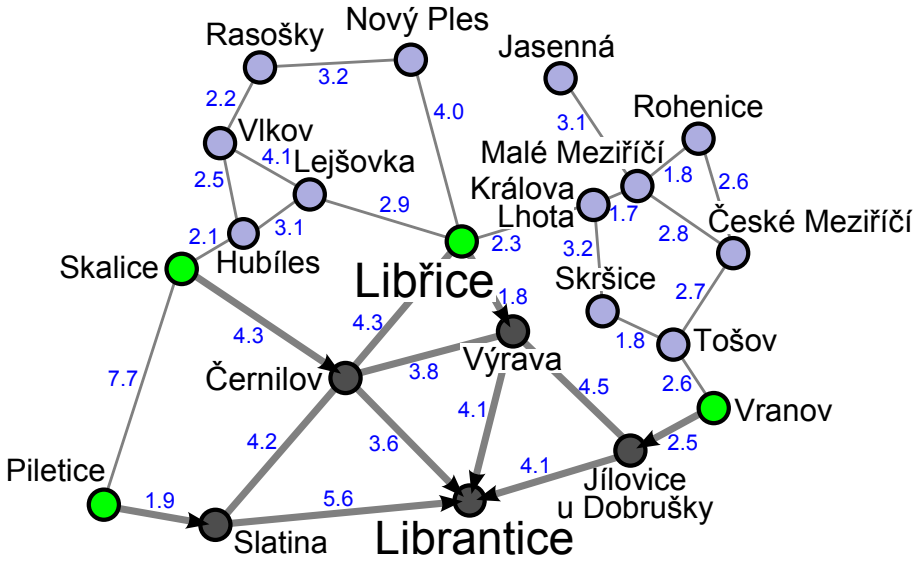
The shortest path from Librantice to Libřice?

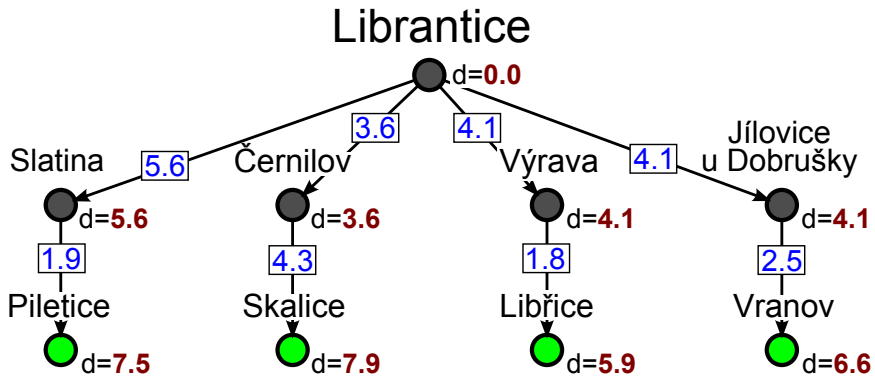




Dijkstra's Algorithm : Example

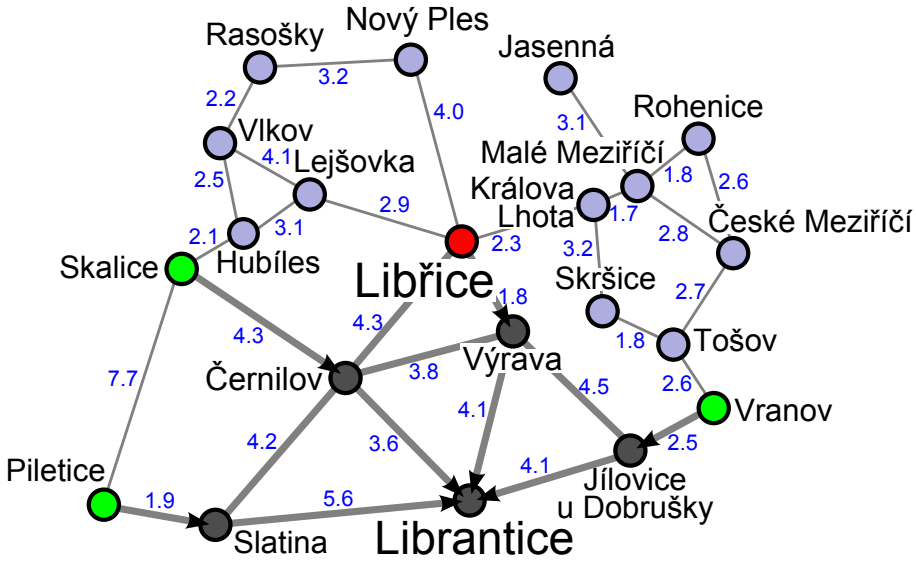
The shortest path from Librantice to Libřice?

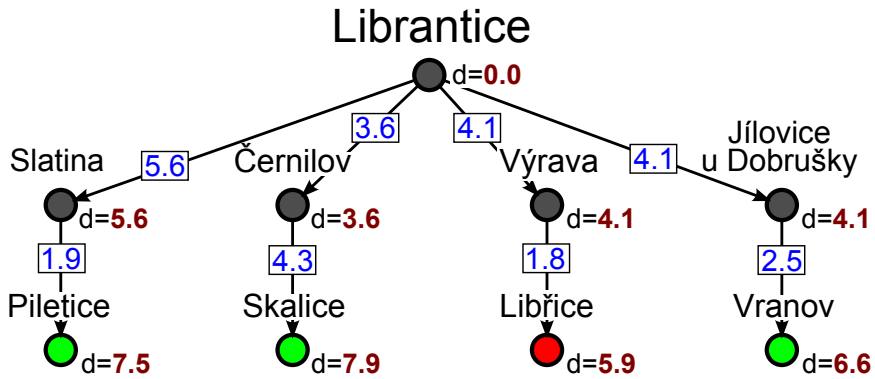




Dijkstra's Algorithm : Example

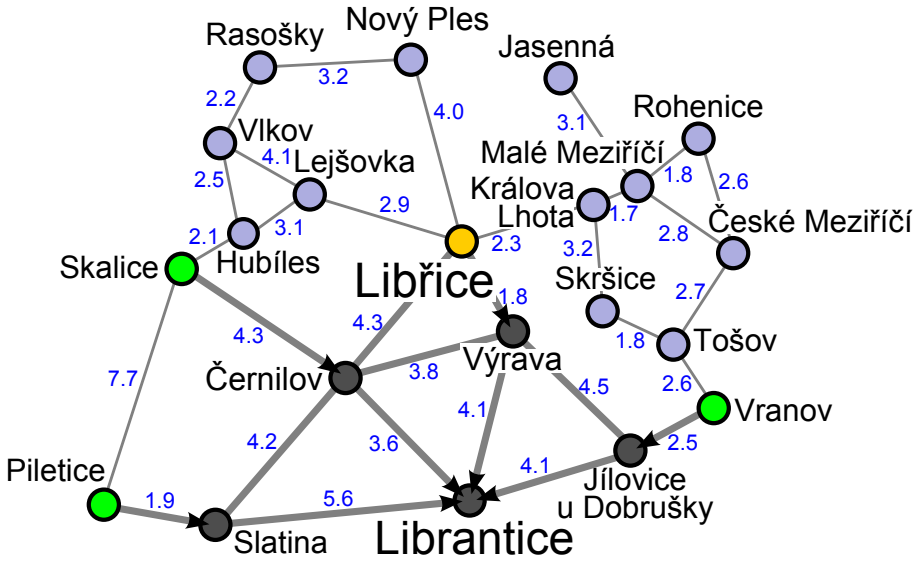
The shortest path from Librantice to Libřice?

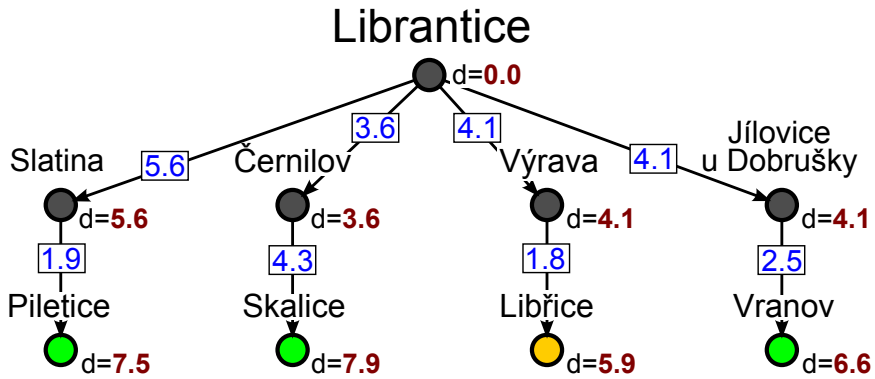




Dijkstra's Algorithm : Example

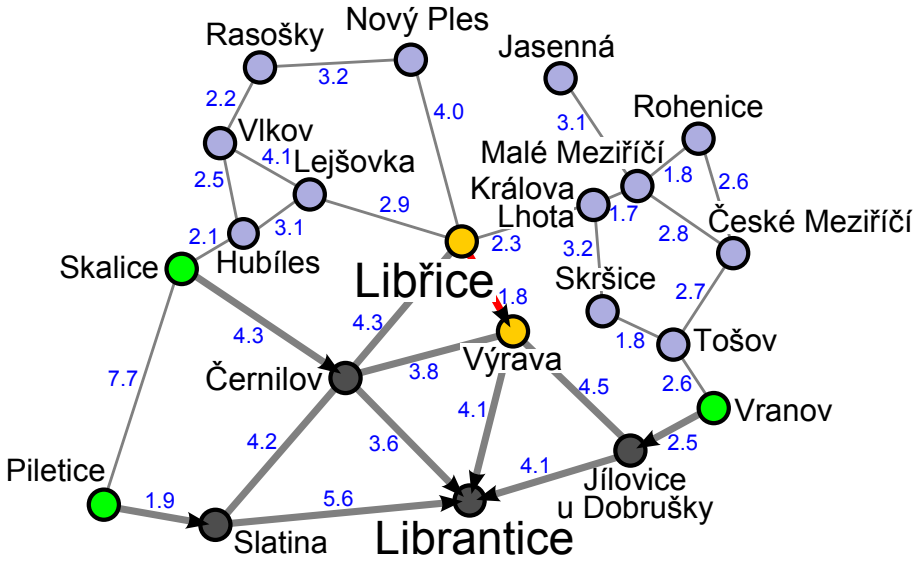
The shortest path from Librantice to Libřice?

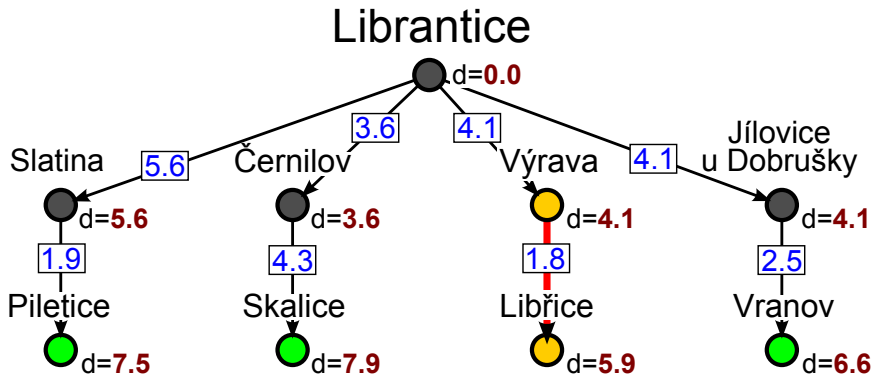




Dijkstra's Algorithm : Example

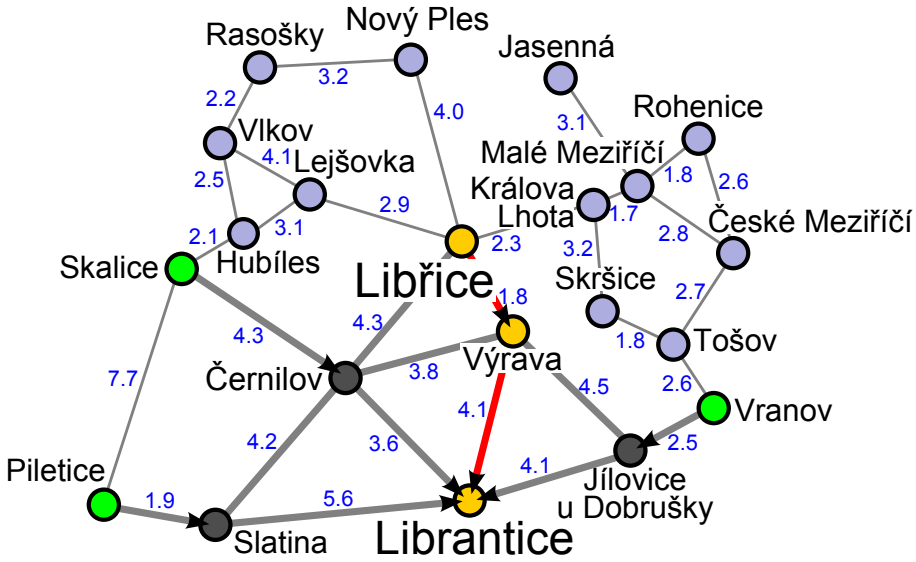
The shortest path from Librantice to Libřice?

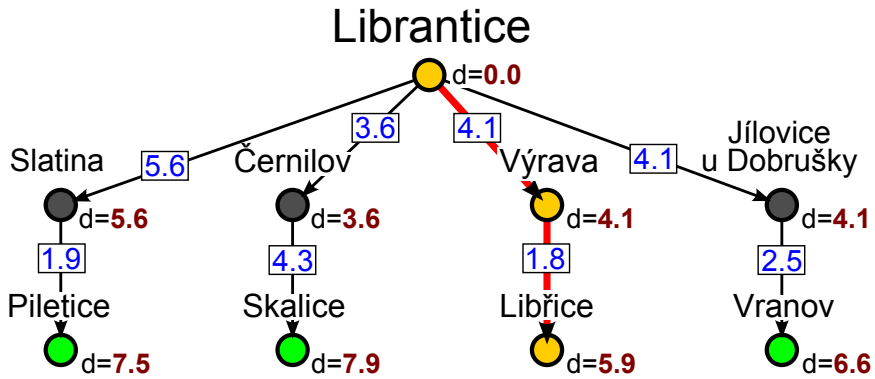




Dijkstra's Algorithm : Example

The shortest path from Librantice to Libřice?





Algorithm 1 Dijkstra

```
 1: Q ← init_priority_queue()
 2: dist ← init_table(); prev ← init_table()
 3: for all s ∈ S \ {init} do
 4:     enqueue(Q, s, ∞)
 5:     dist[s] ← ∞; prev[s] ← null
 6: end for
 7: enqueue(Q, init, 0); dist[init] ← 0
 8: while ¬empty(Q) do
 9:     x ← dequeue(Q)
10:     if x ∈ G then
11:         return reconstruct_path(prev, x)
12:     end if
13:     for all y ∈ neighbors(x) do
14:         d' ← dist[x] + c((x, y))
15:         if d' < dist[y] then
16:             dist[y] ← d'; prev[y] ← x
17:             update_key(Q, y, d')
18:         end if
19:     end for
20: end while
```

Heuristic (Informed) Search

- Uninformed search strategies rely only on a goal function.
- Informed search uses heuristic functions, that are specific to the problem and whose purpose is to guide the search towards the goal.
- Heuristics utilize some specific knowledge of the problem domain to estimate the quality or potential of partial solution.
- Heuristic algorithms are more efficient since a good heuristic function can dramatically reduce the amount of expanded nodes.
- Heuristics does not always guarantee optimal solution, however it guarantees to find a good solution in a reasonable time.

Heuristic Function

Definition (Heuristic Function)

Heuristic is any function

$$h\colon S \to \mathbb{R}_0^+,$$

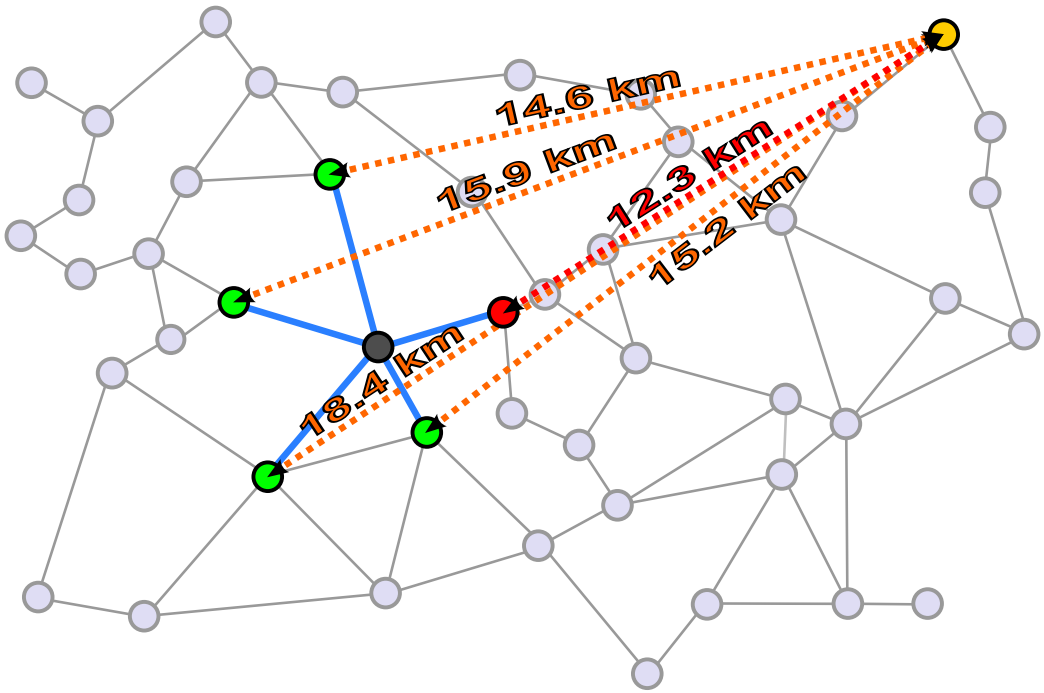
so that $h(s)$ assigns to each state $s \in S$ an estimate of the least cost path to the closest goal state $s_g \in G$ and $\forall s_g \in G : h(s_g) = 0$.

Definition (State Space with Heuristic)

A state-space with heuristic is a tuple $\langle S, A, \mathcal{I}, G, c, h \rangle$, such as (S, A) is a directed graph with a set of nodes S representing states and a set of edges A representing actions, $\mathcal{I}$ is a set of initial states, G is a set of goal states, $c(a)$ is a cost function assigning a non negative cost to each action a and $h(s)$ is a heuristic function assigning an estimate of the least cost path to the closest goal state from each state s.

Example: Route Finding

- In the route finding problem we can use the information about geographical location between the cities.
- As a heuristic we can use the air distance to the goal city, which is the shortest possible distance.



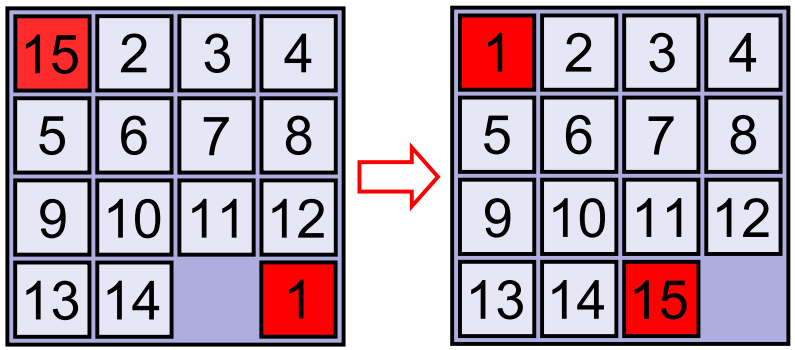
Example: Sliding Puzzles

- In the sliding puzzles problem we can use the information about the goal configuration.
- A heuristic can be manhattan distance for each tile to final position.

Manhattan distance heuristic

$$h(s) = \sum_{i=1}^{N} (|x_i(s) - x_i^*| + |y_i(s) - y_i^*|),$$

where $x_i(s)$ and $y_i(s)$ are the x and y coordinates of tile i in state s, and if x_i^* and y_i^* are the x and y coordinates of tile i in the goal state.



$$h(s) = 6 + 5 = 11$$

Greedy Search

- Algorithm always expands the node with the best (lowest) heuristic value

$$s^* = \arg\min_{s \in OPEN} h(s)$$

- Greedy search chooses the node that appears to be the closest to the goal since it assumes that this leads quickly to a solution.
- Success of greedy search depends on quality of heuristic function.

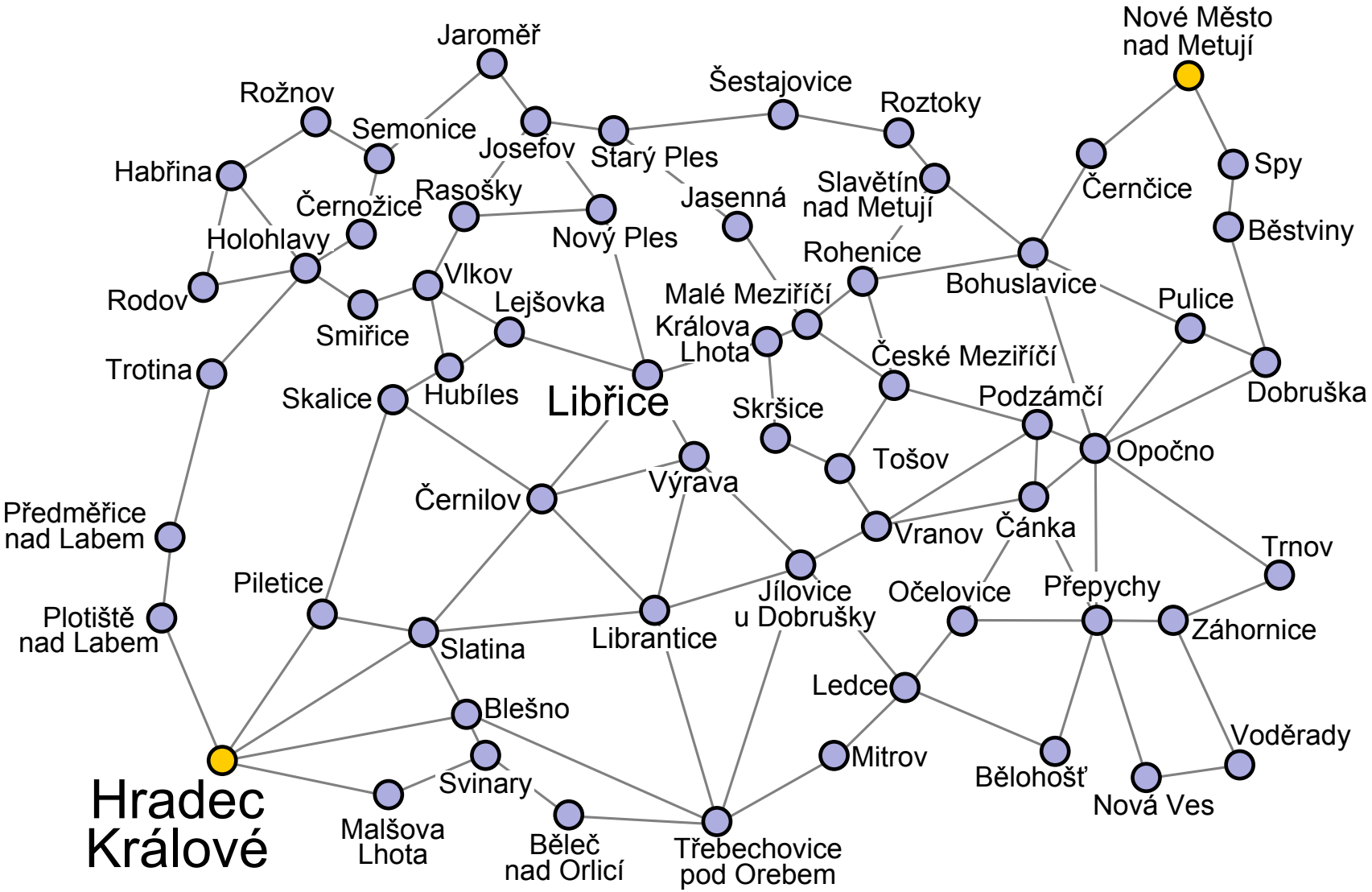
Complete: No

Optimal: No

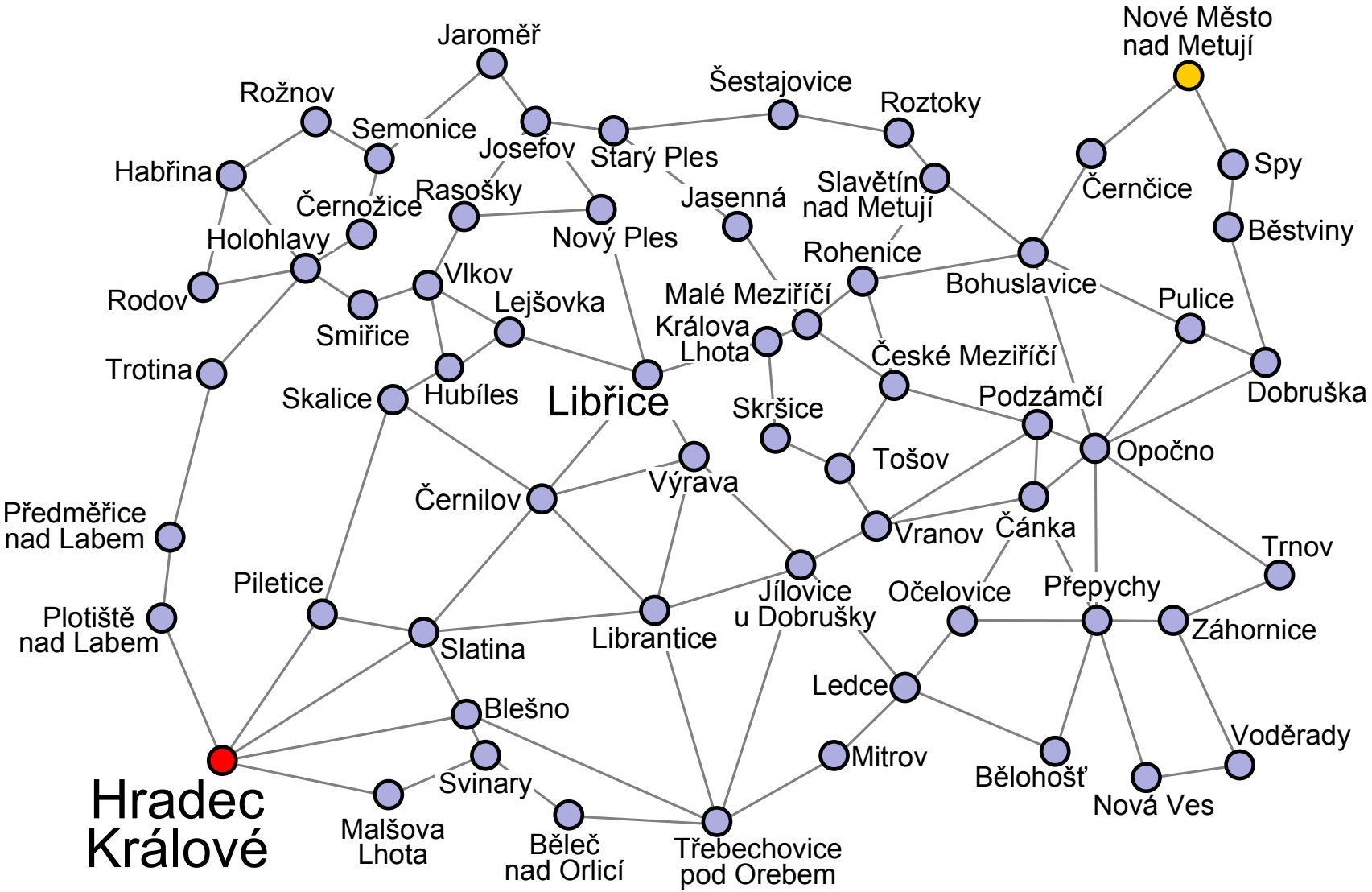
Time: $O(b^m)$

Space: $O(b^m)$

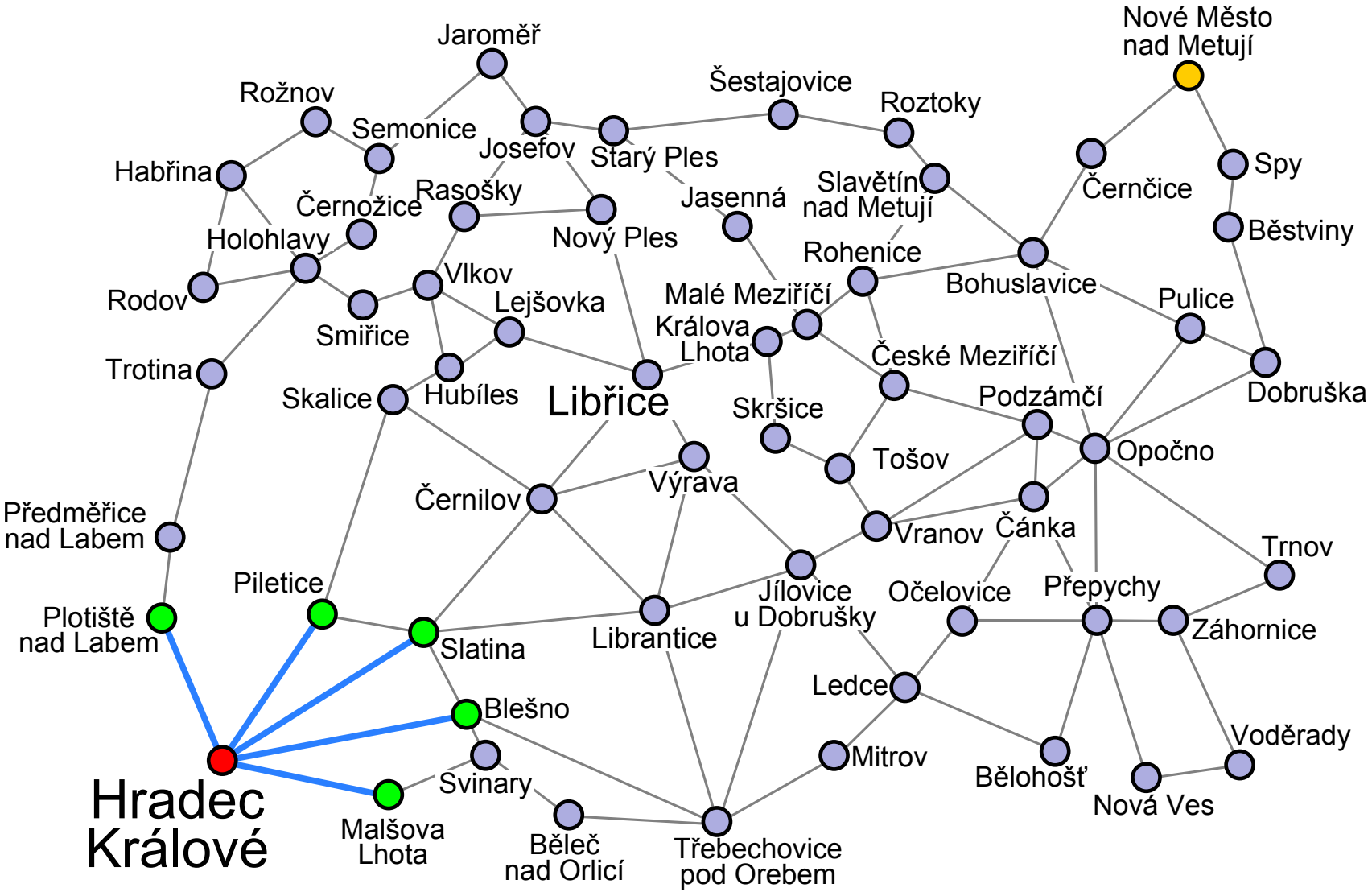
Greedy Search: Example



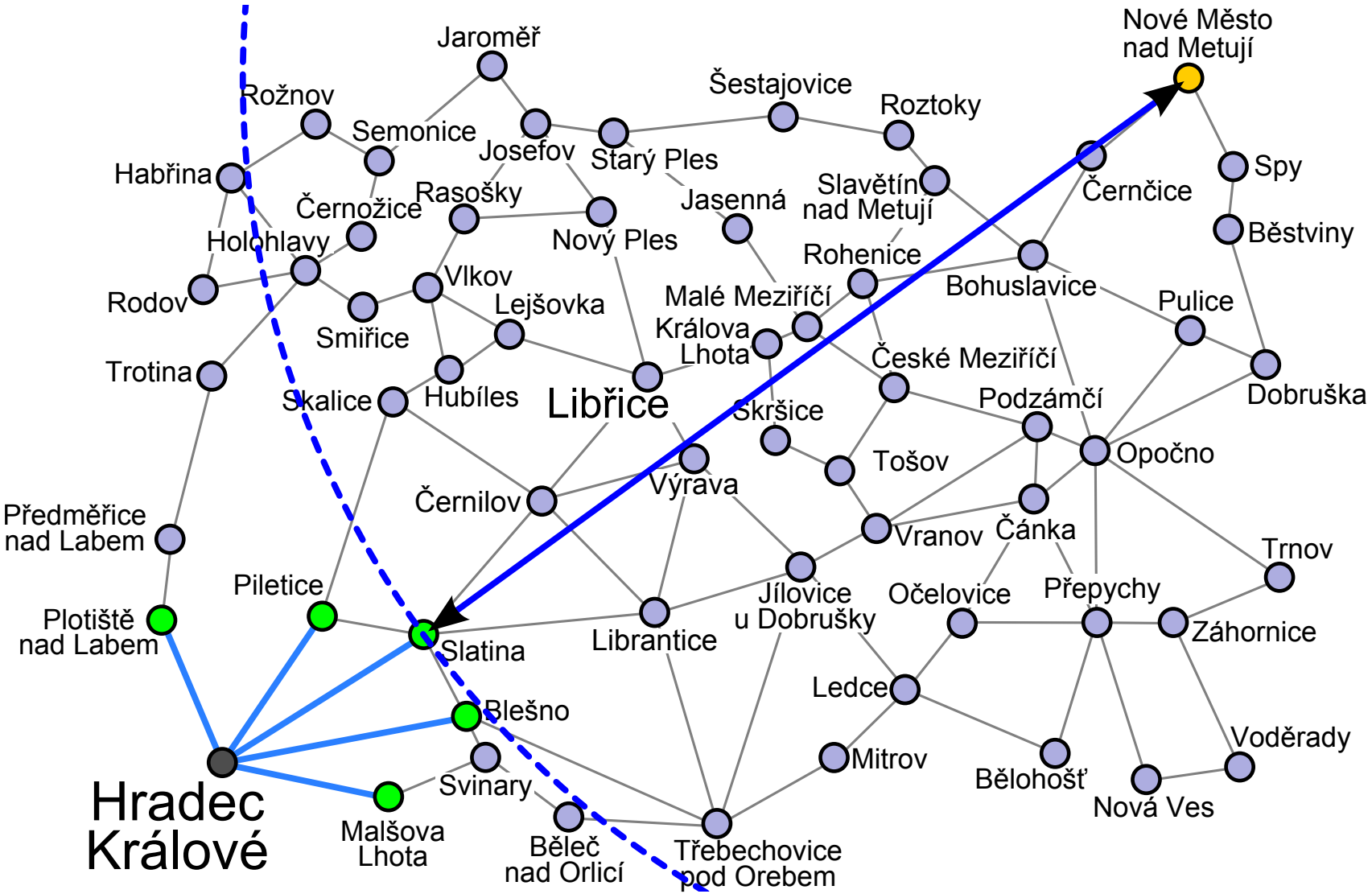
Greedy Search: Example



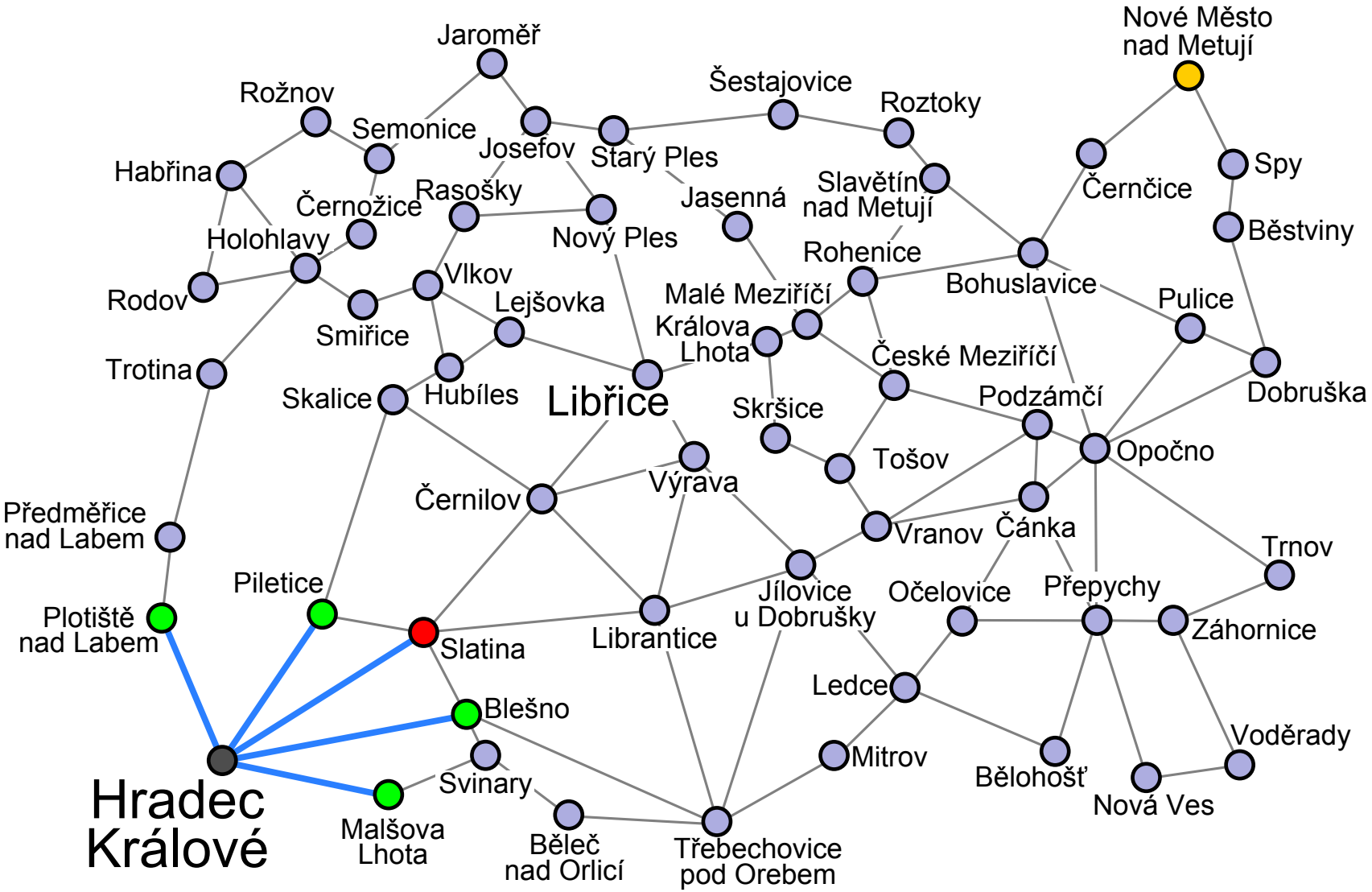
Greedy Search: Example



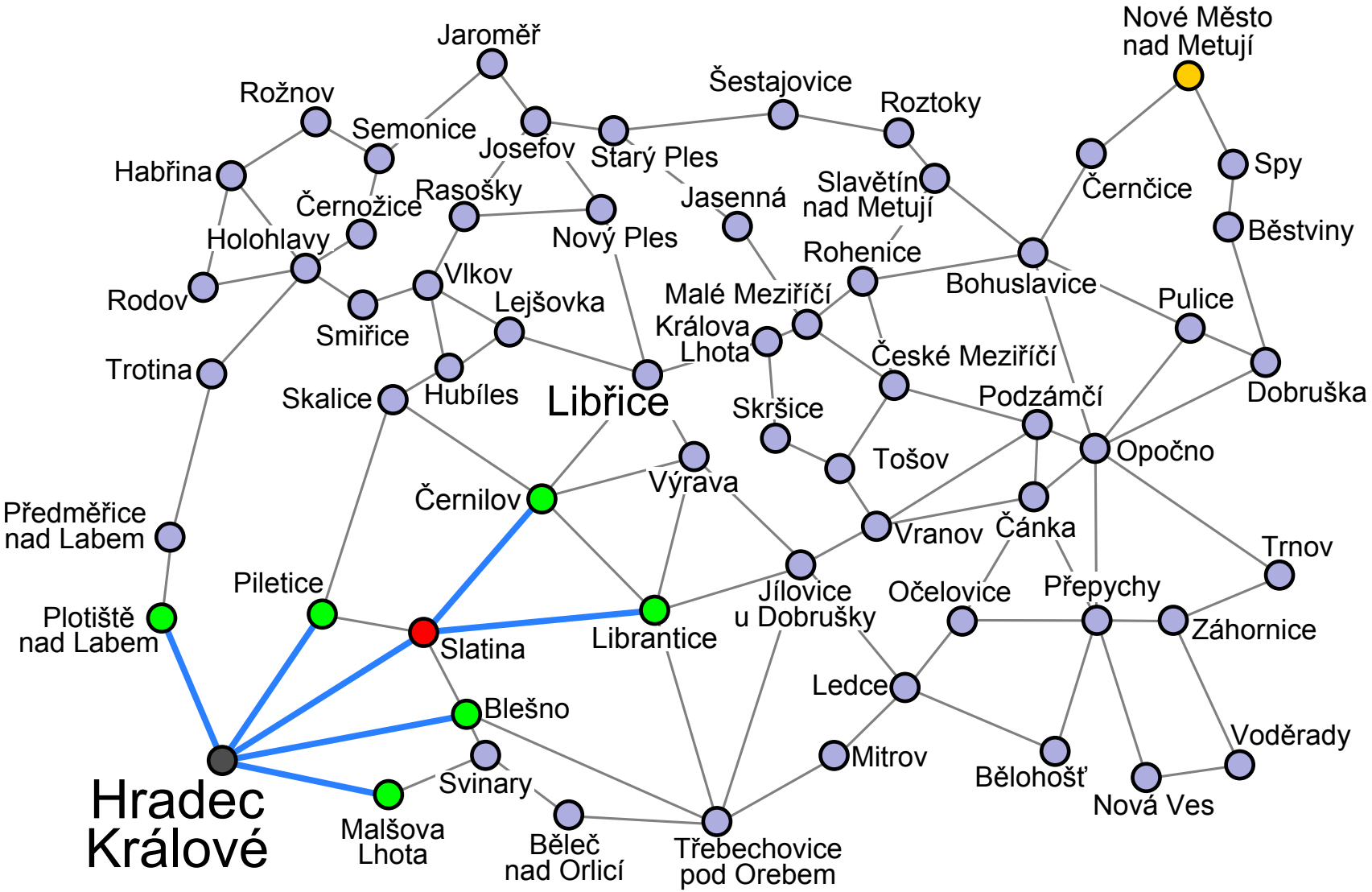
Greedy Search: Example



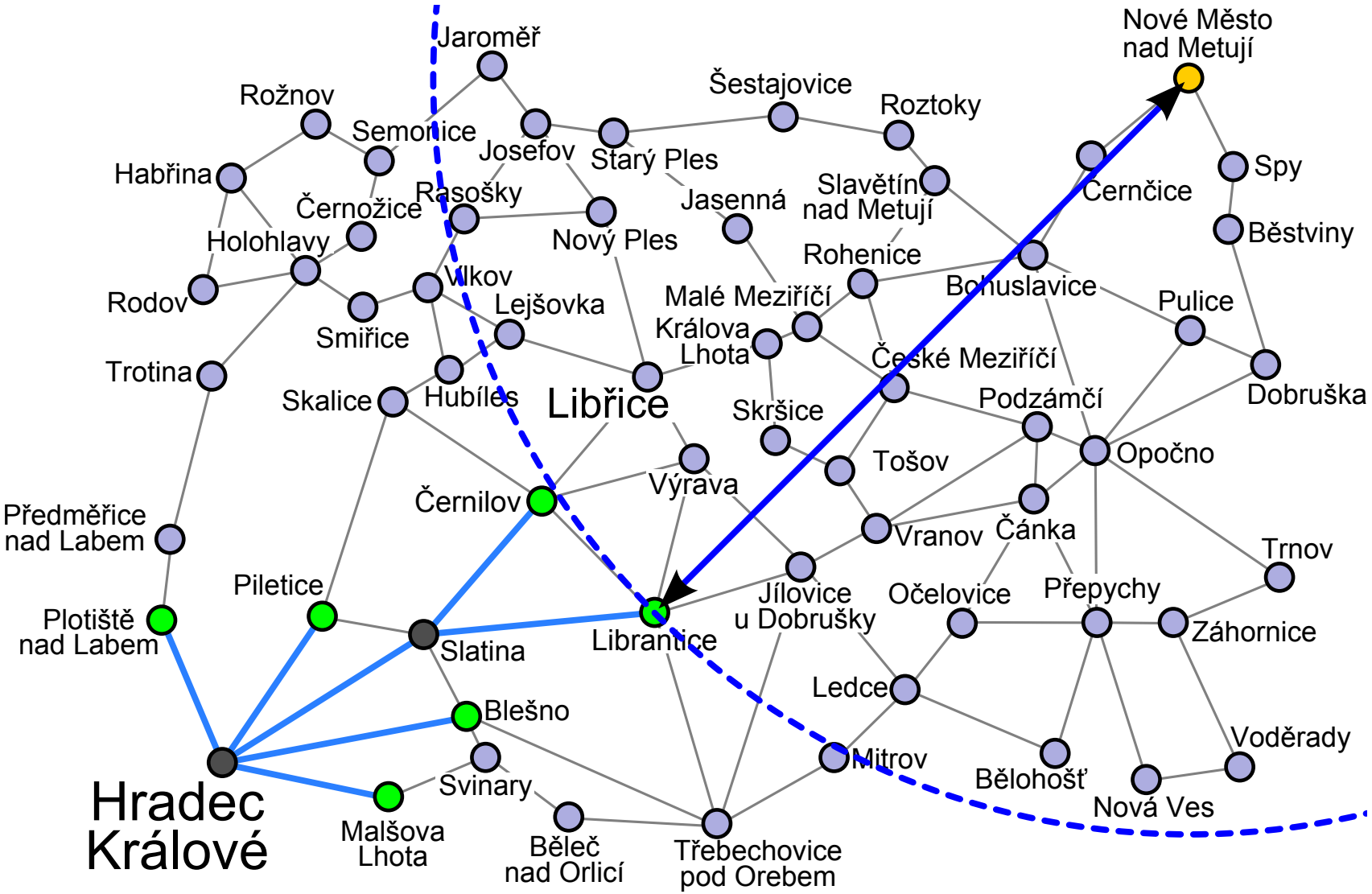
Greedy Search: Example



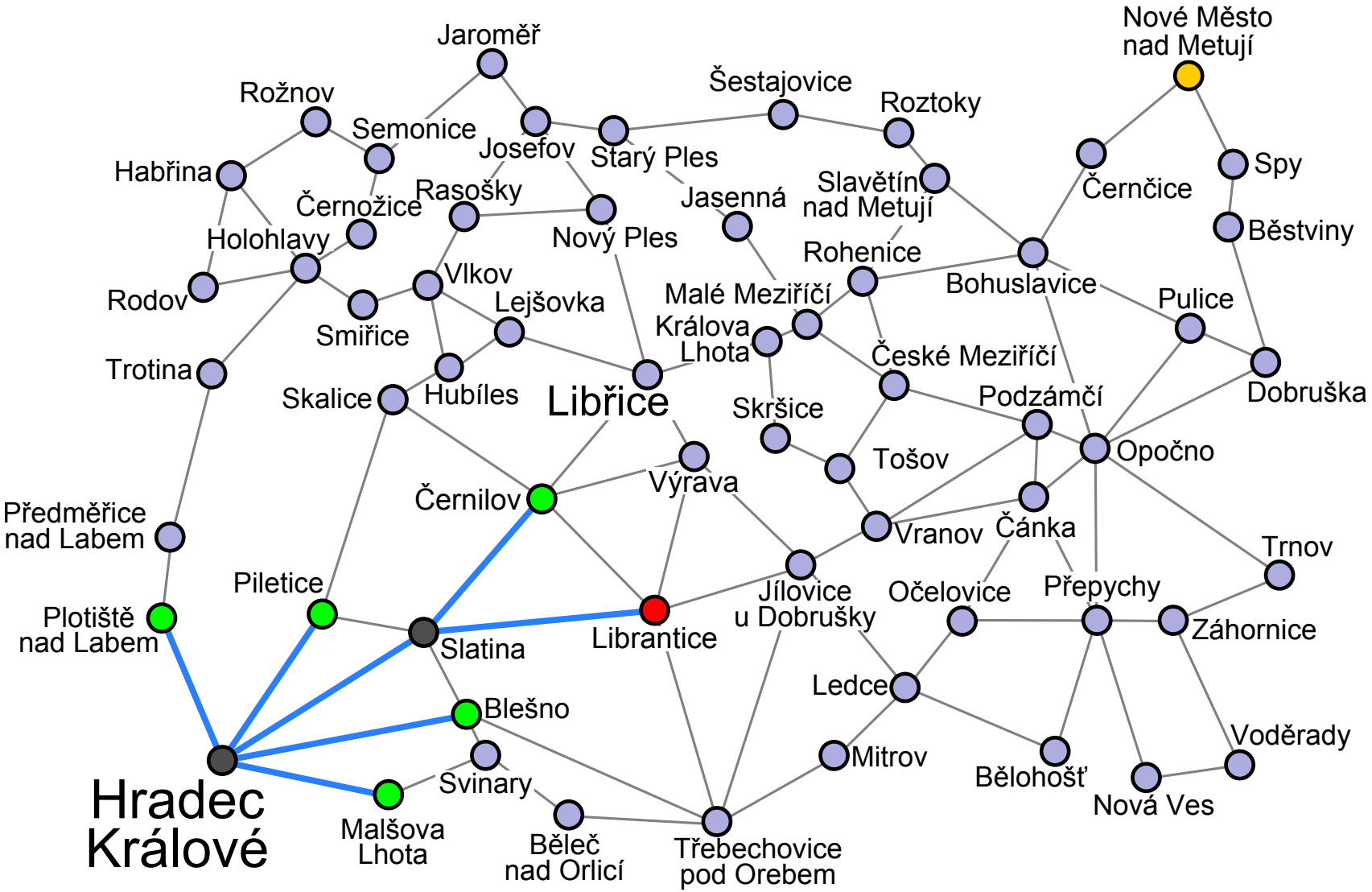
Greedy Search: Example



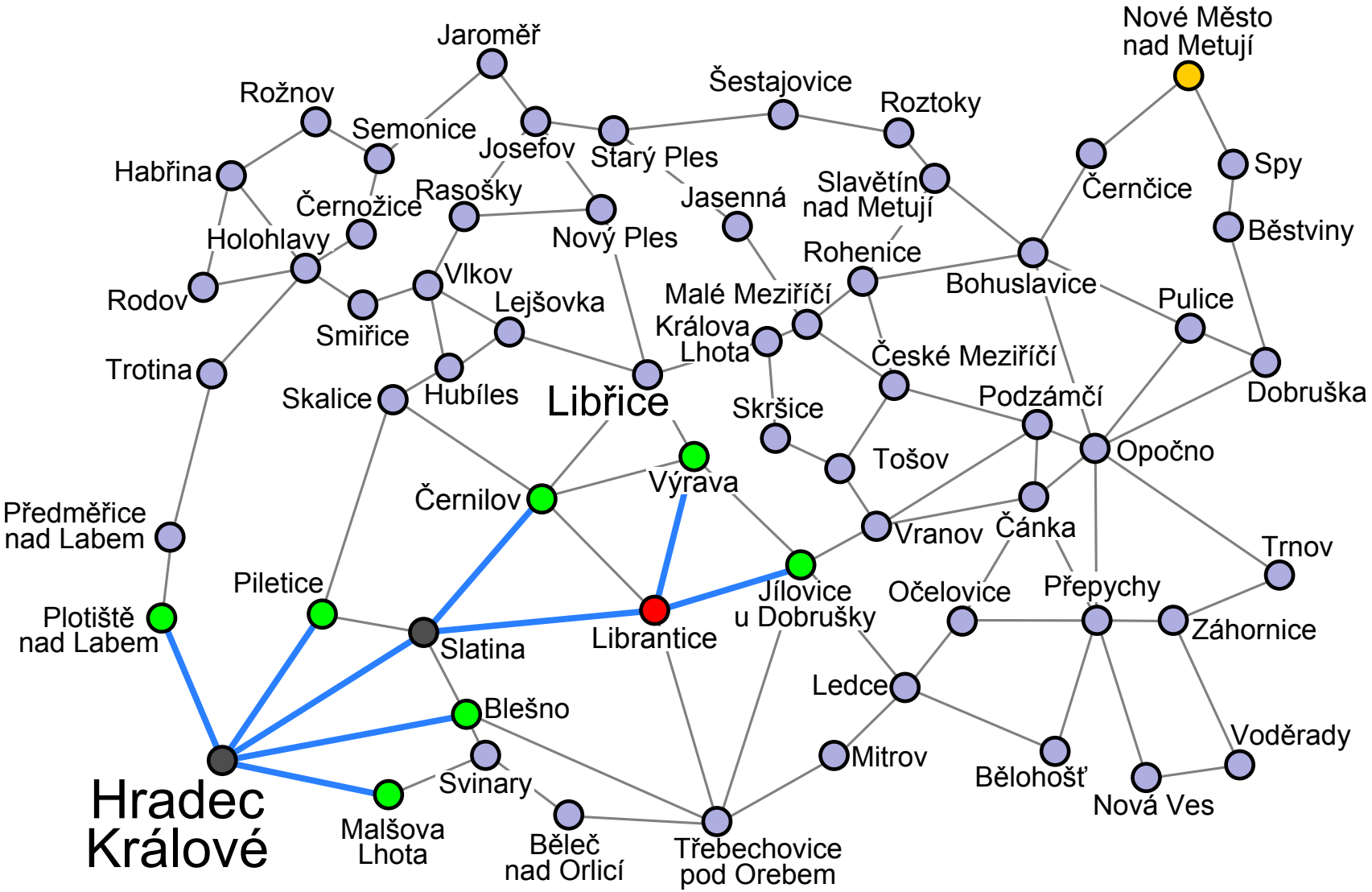
Greedy Search: Example



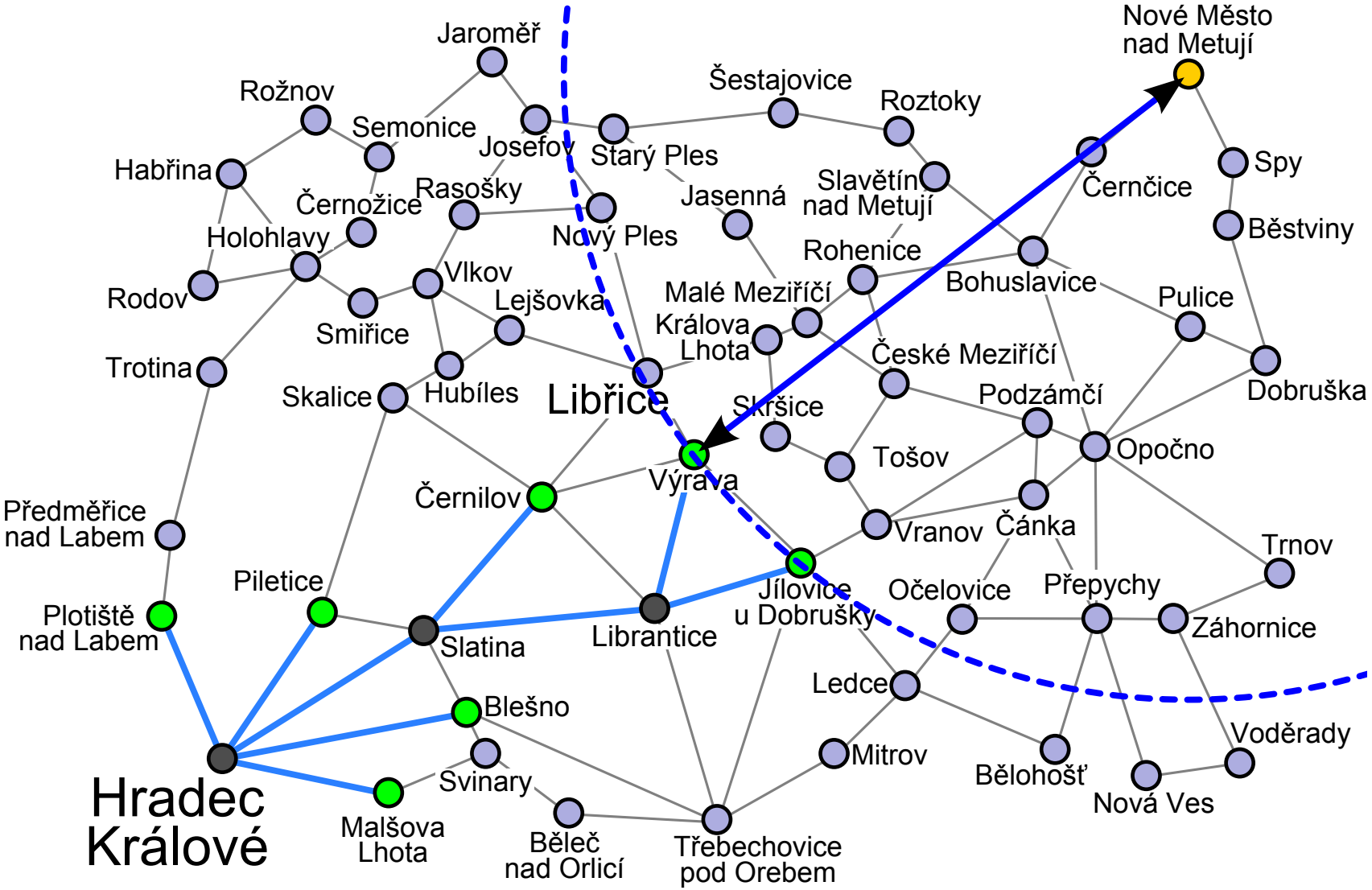
Greedy Search: Example



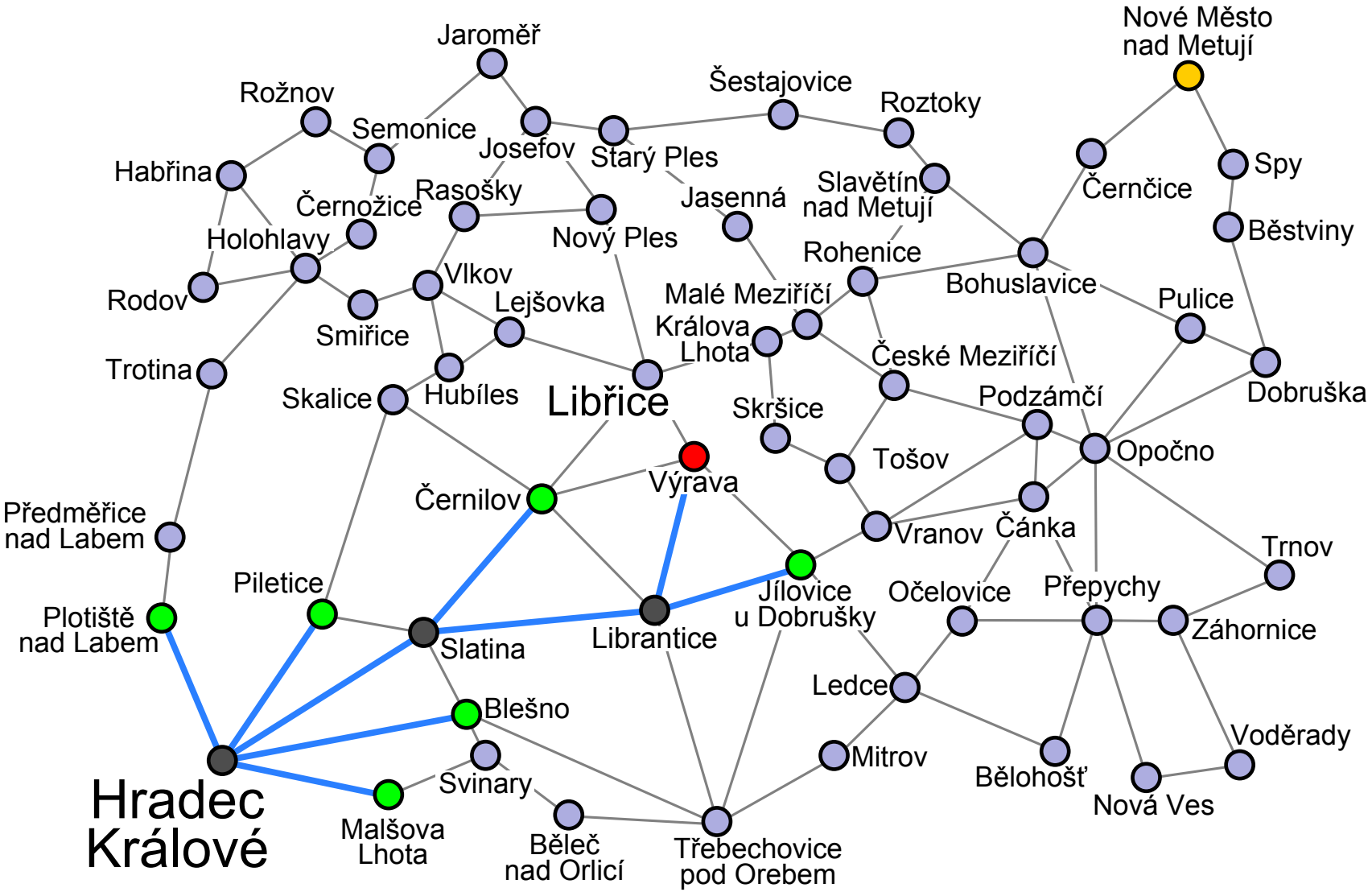
Greedy Search: Example



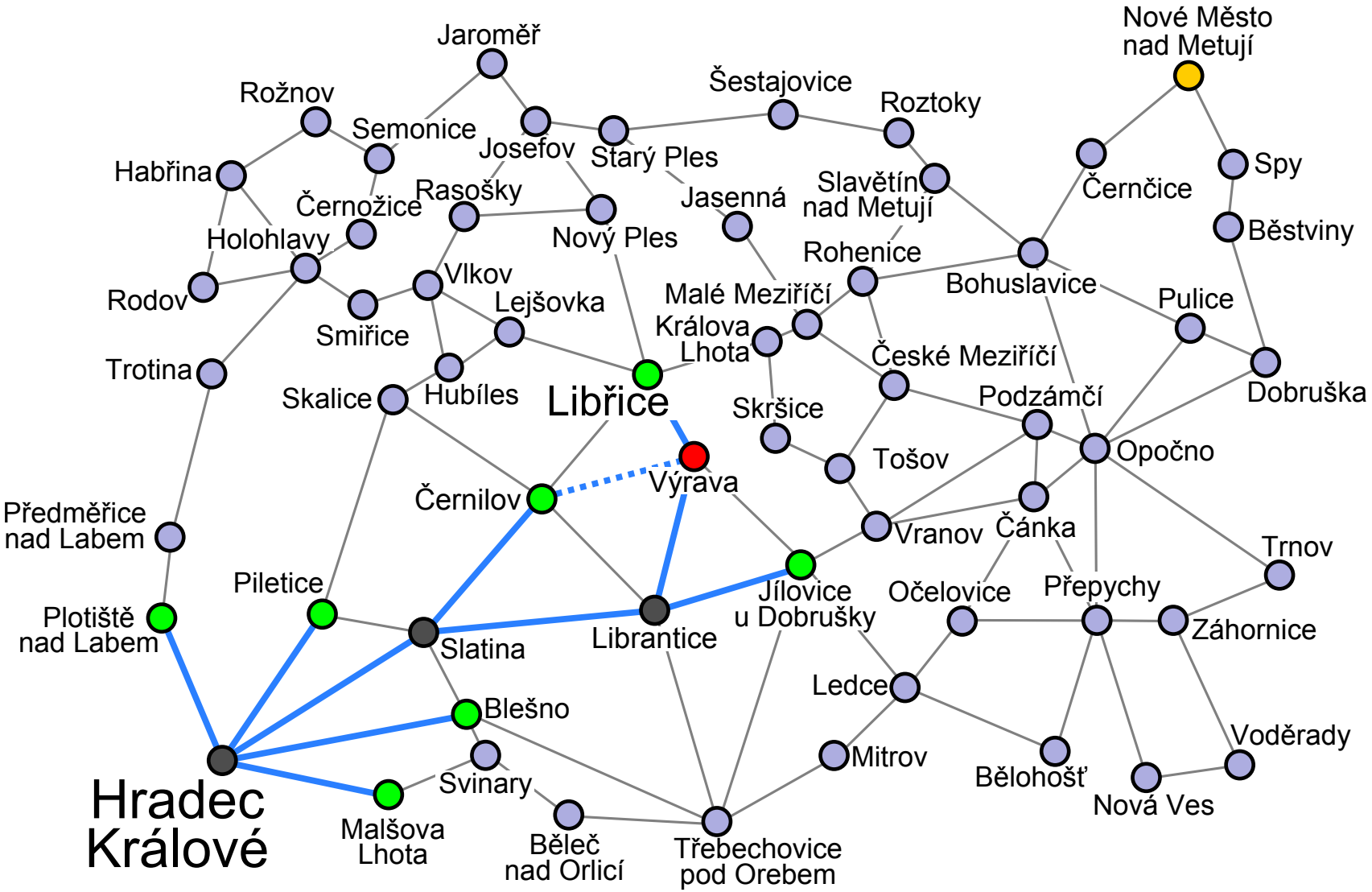
Greedy Search: Example



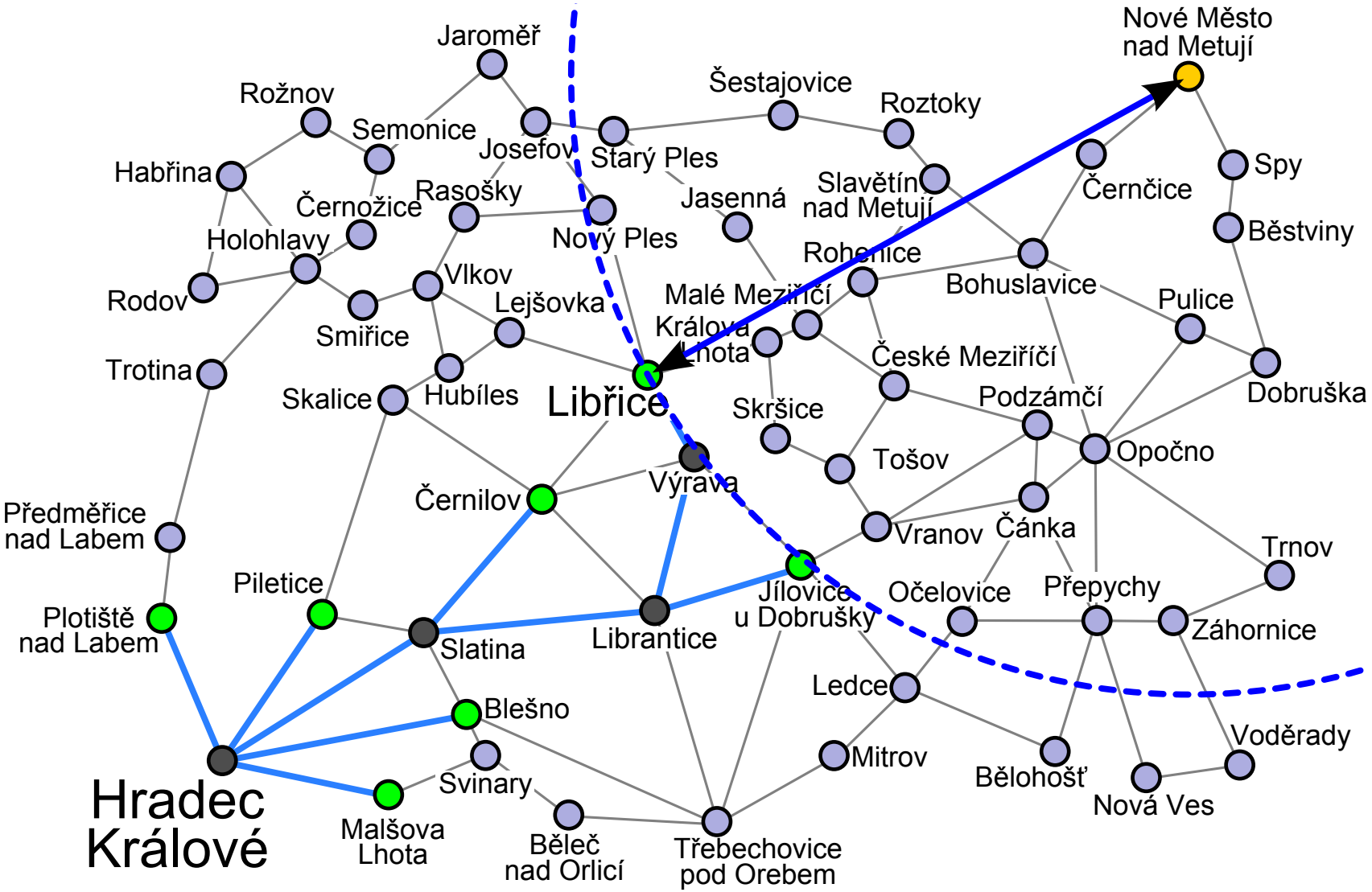
Greedy Search: Example



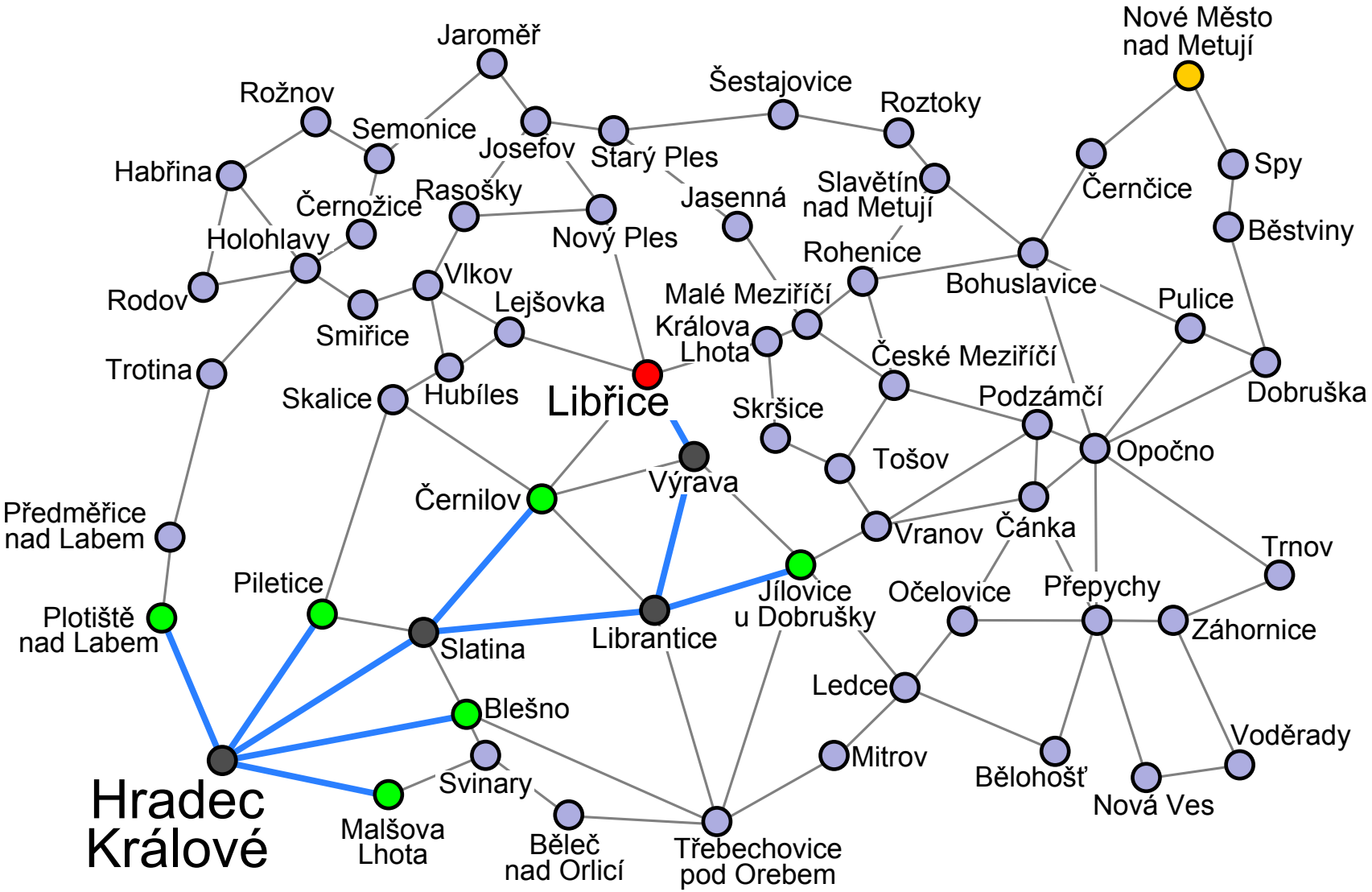
Greedy Search: Example



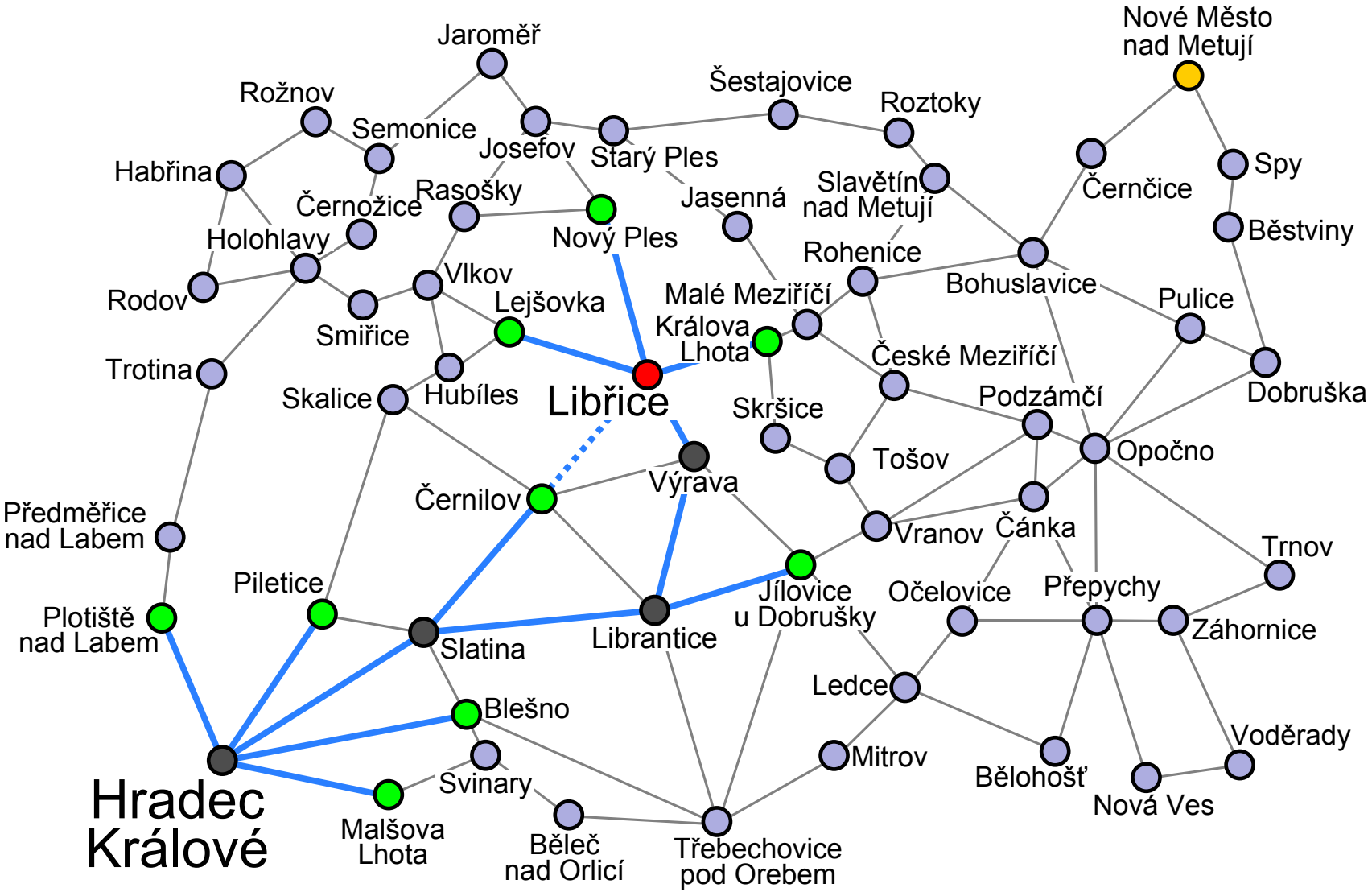
Greedy Search: Example



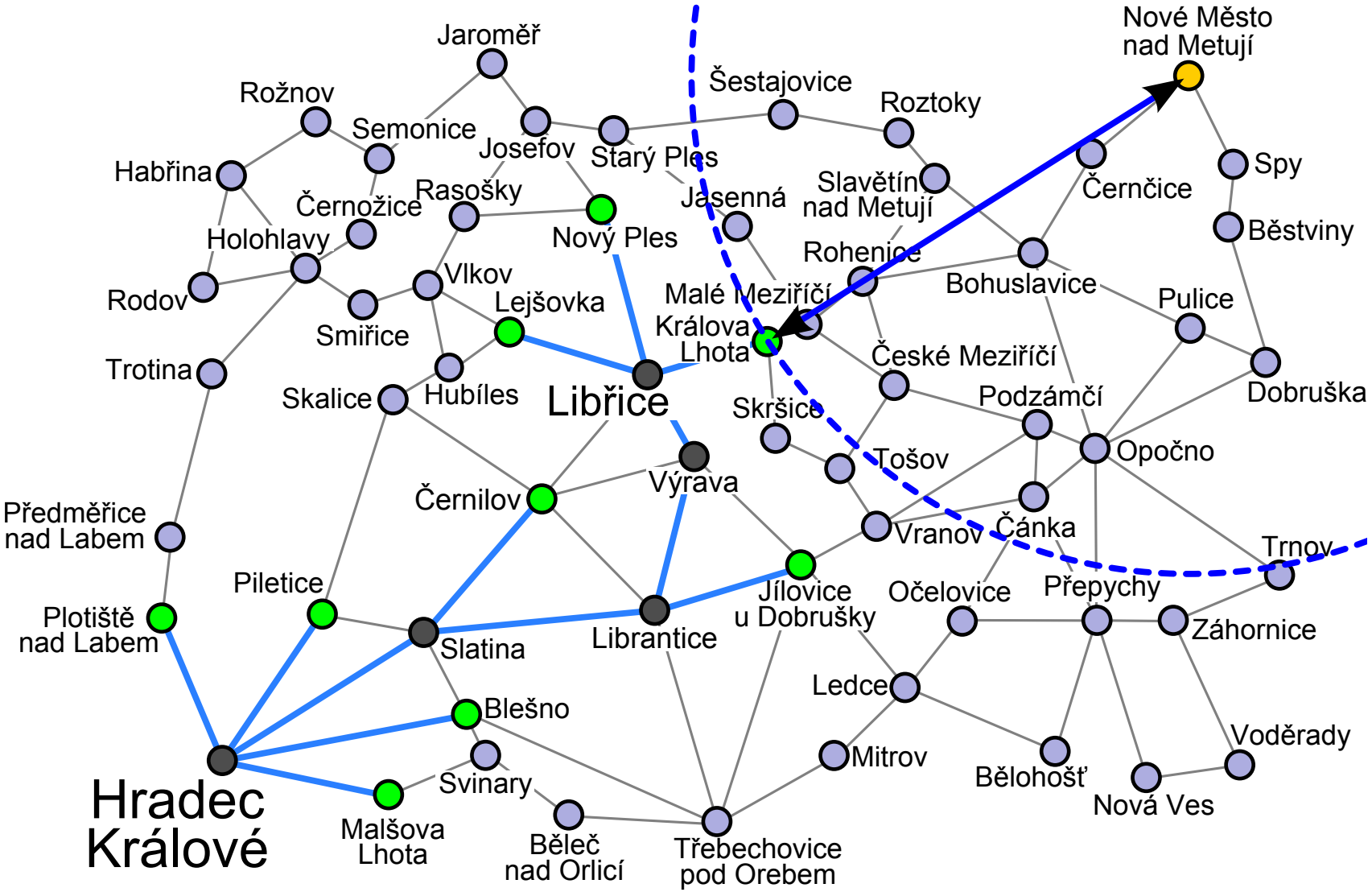
Greedy Search: Example



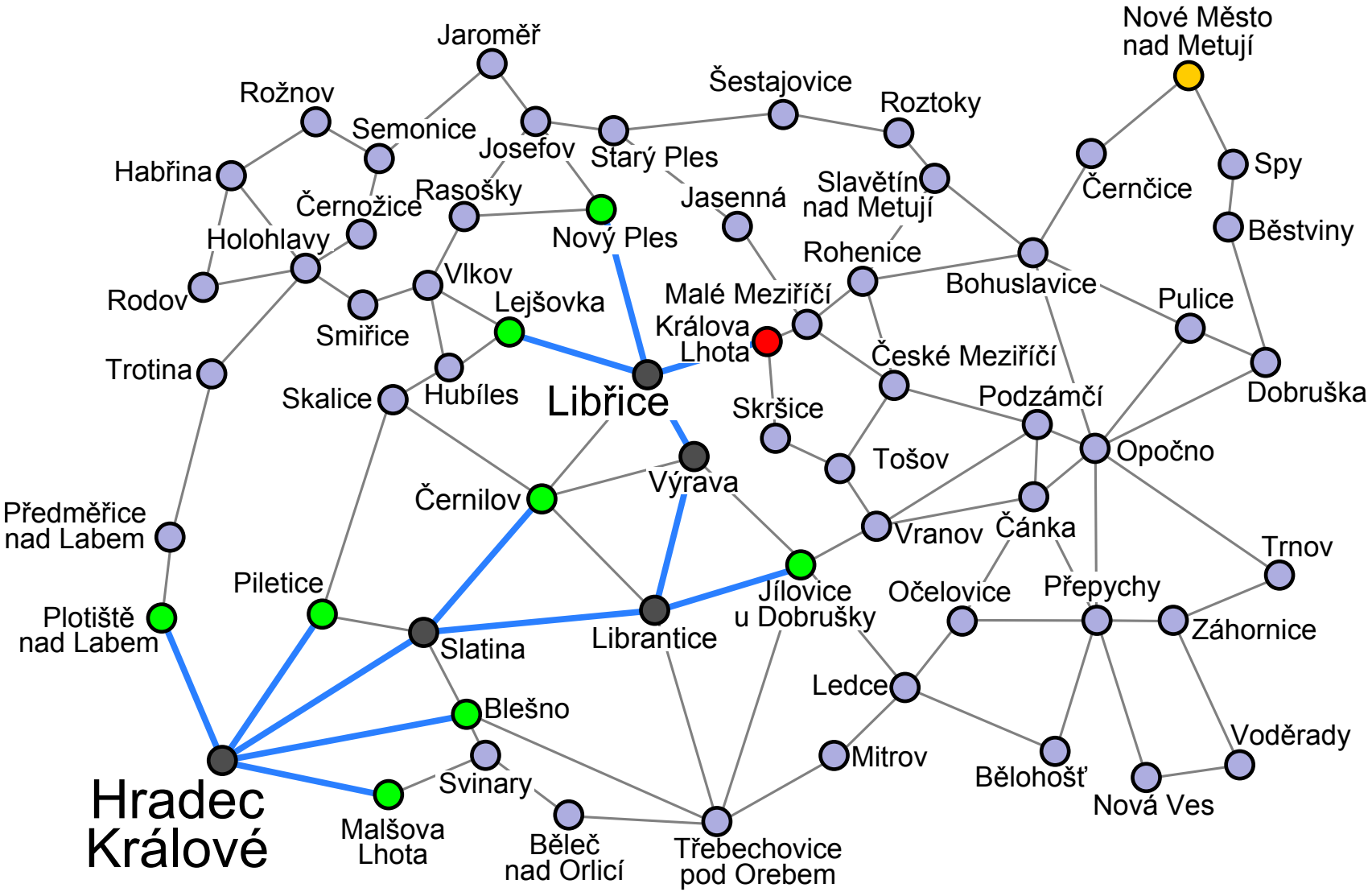
Greedy Search: Example



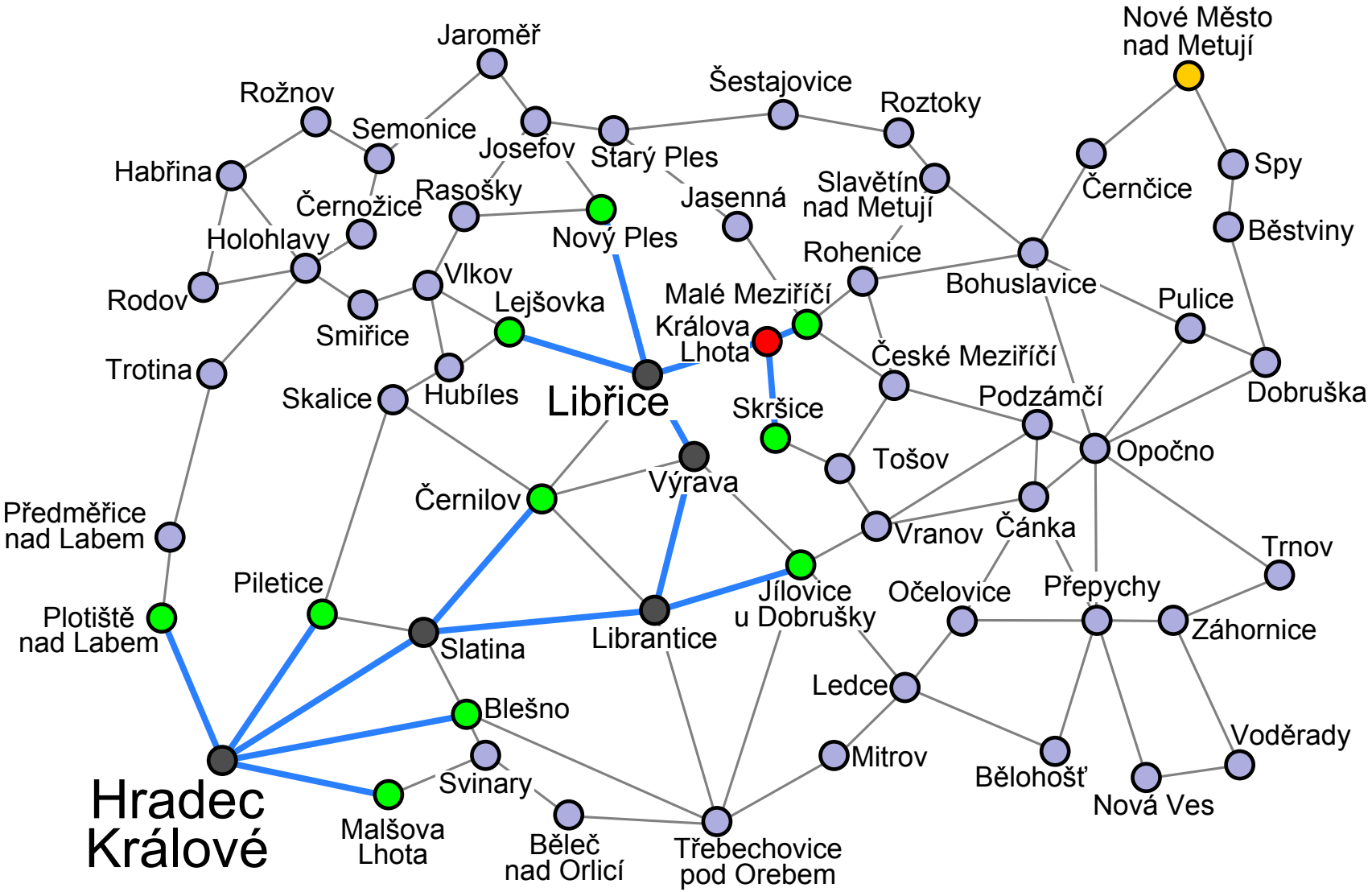
Greedy Search: Example



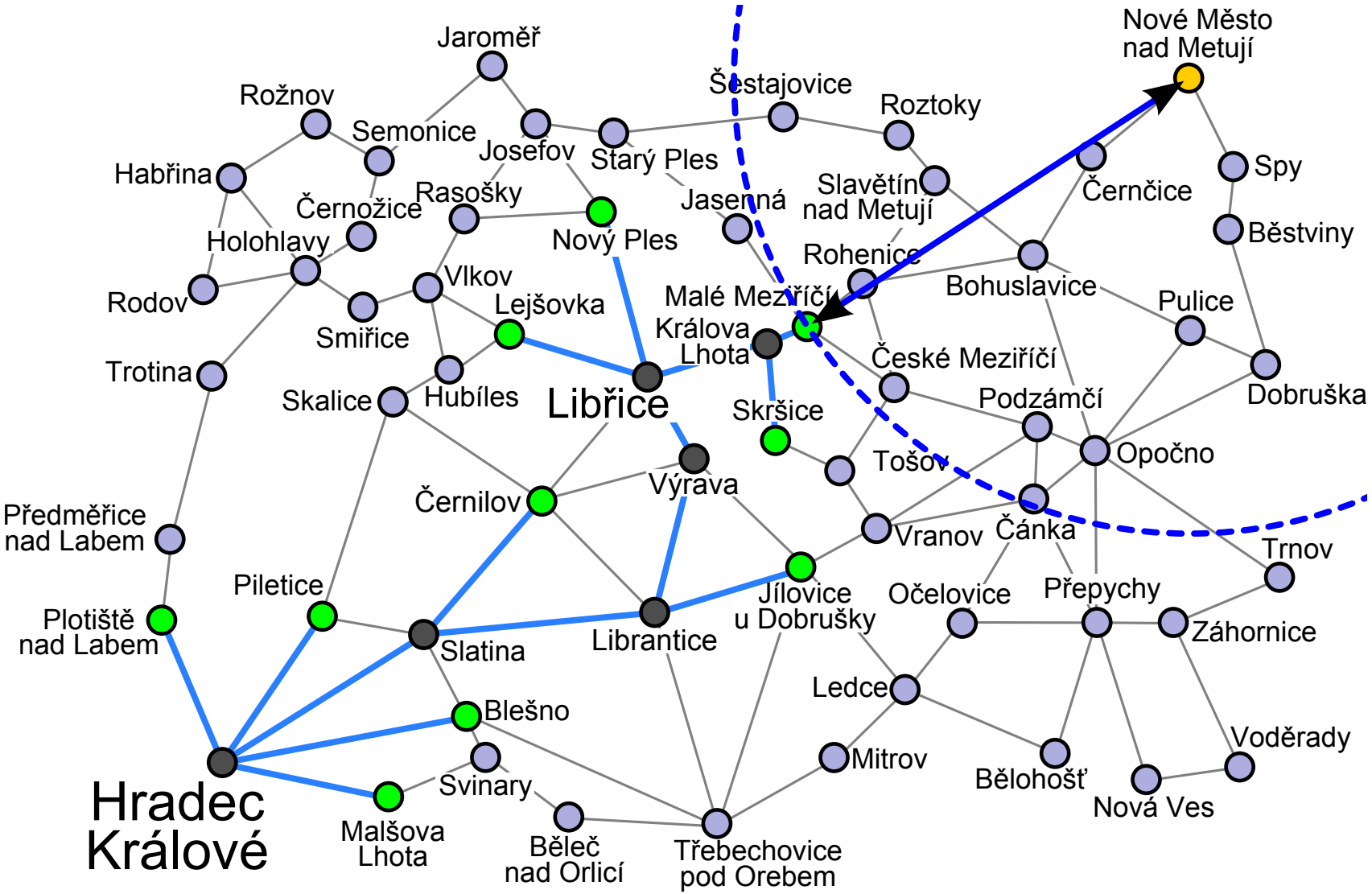
Greedy Search: Example



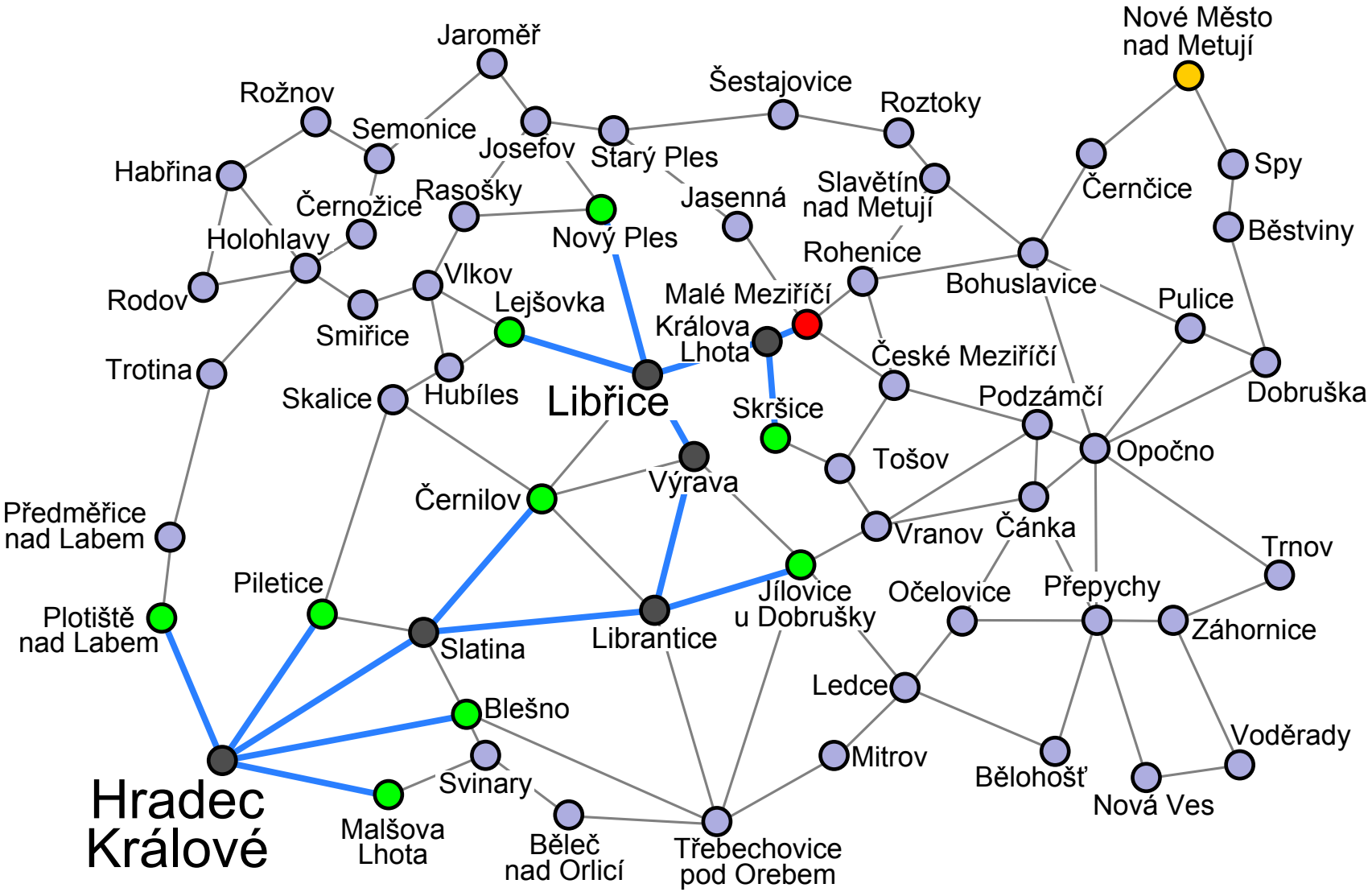
Greedy Search: Example



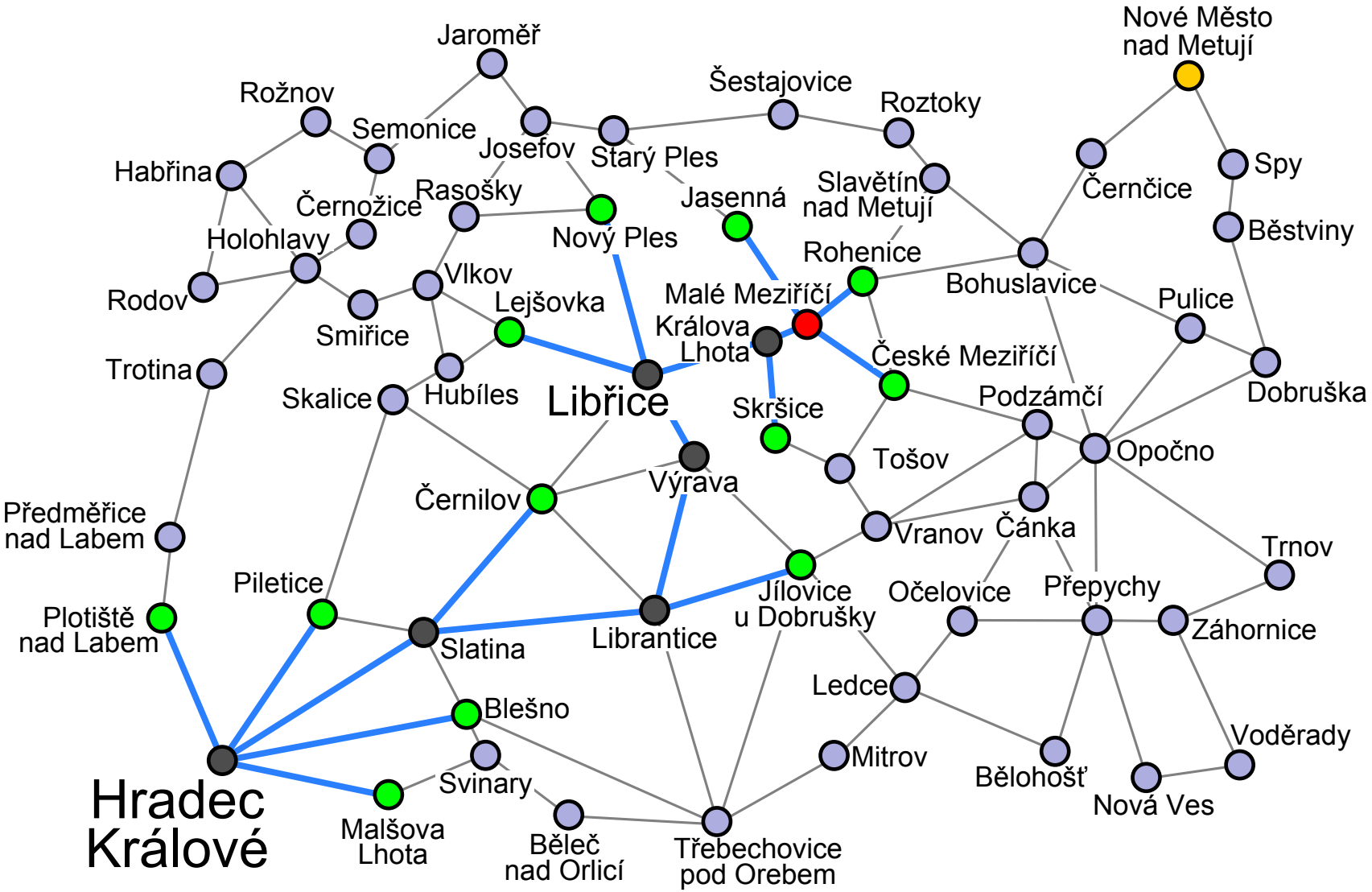
Greedy Search: Example



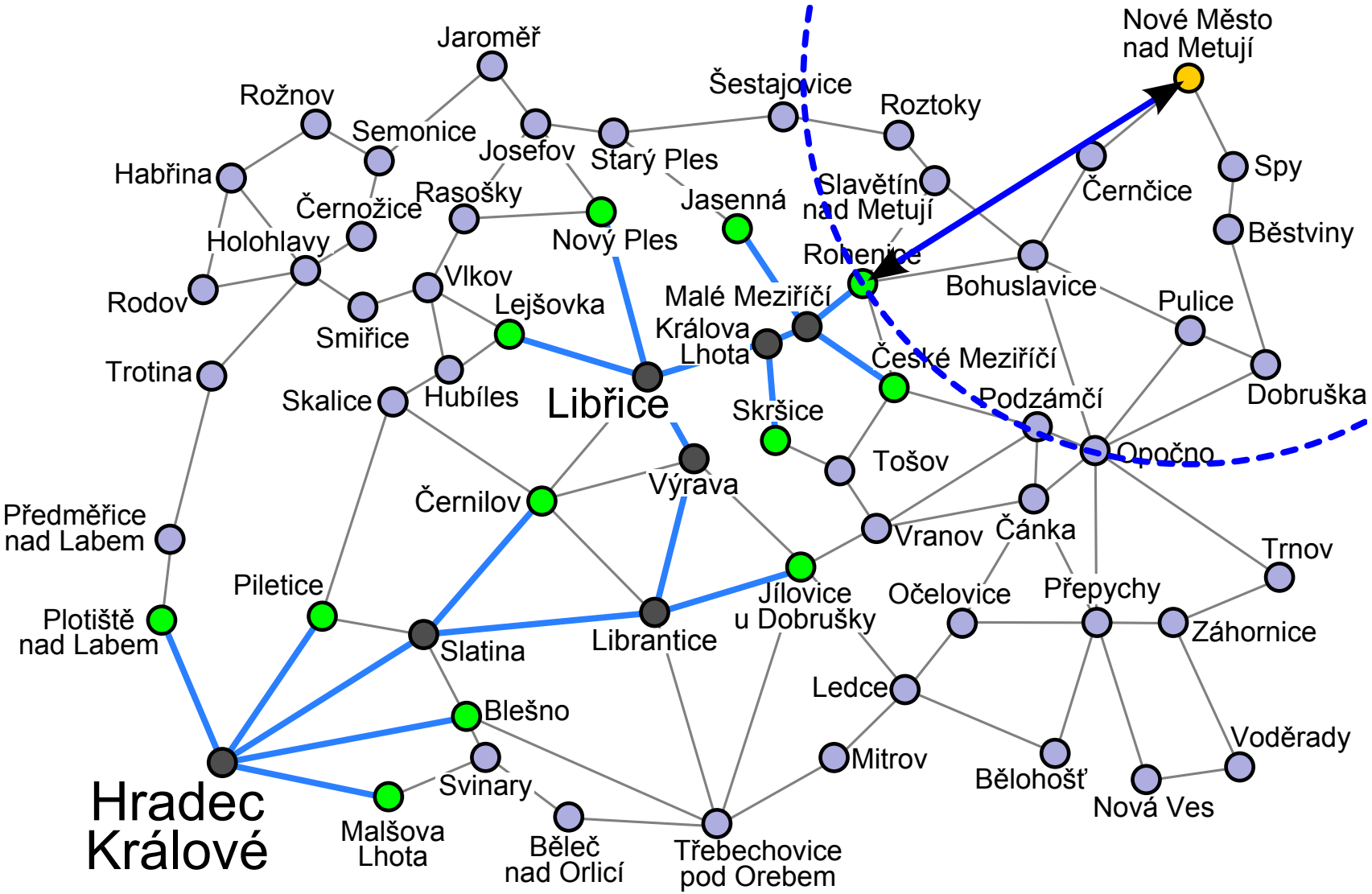
Greedy Search: Example



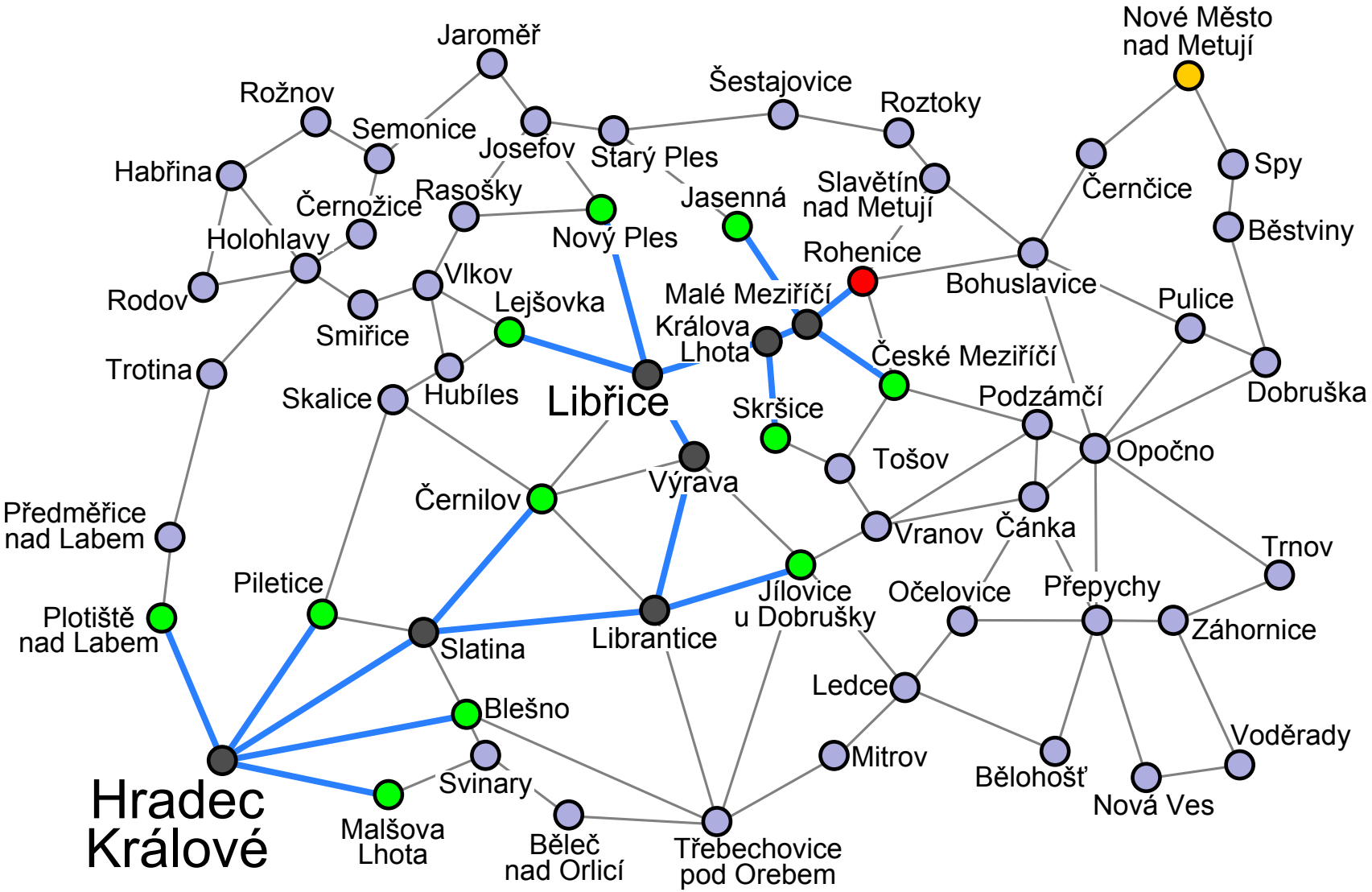
Greedy Search: Example



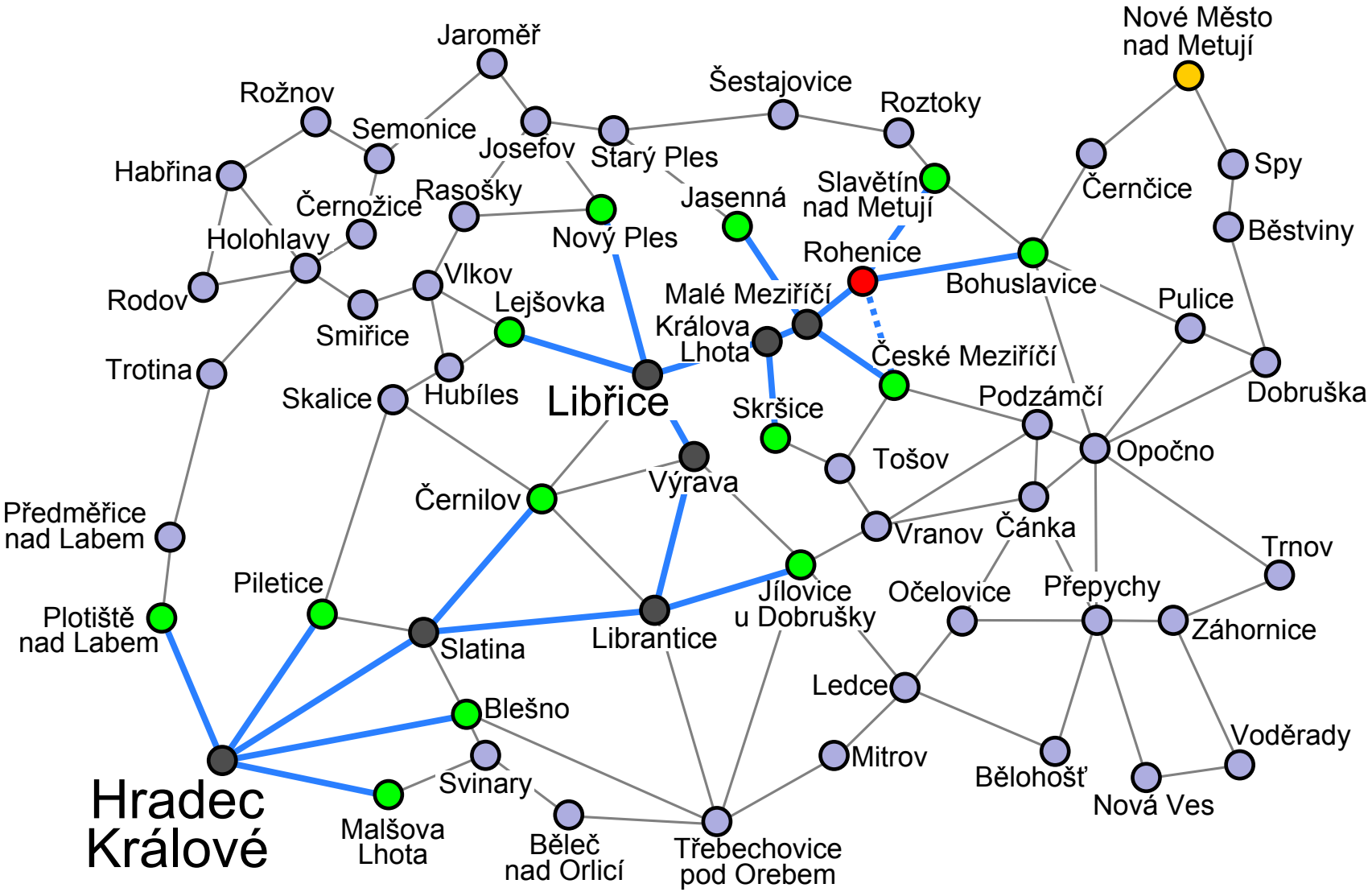
Greedy Search: Example



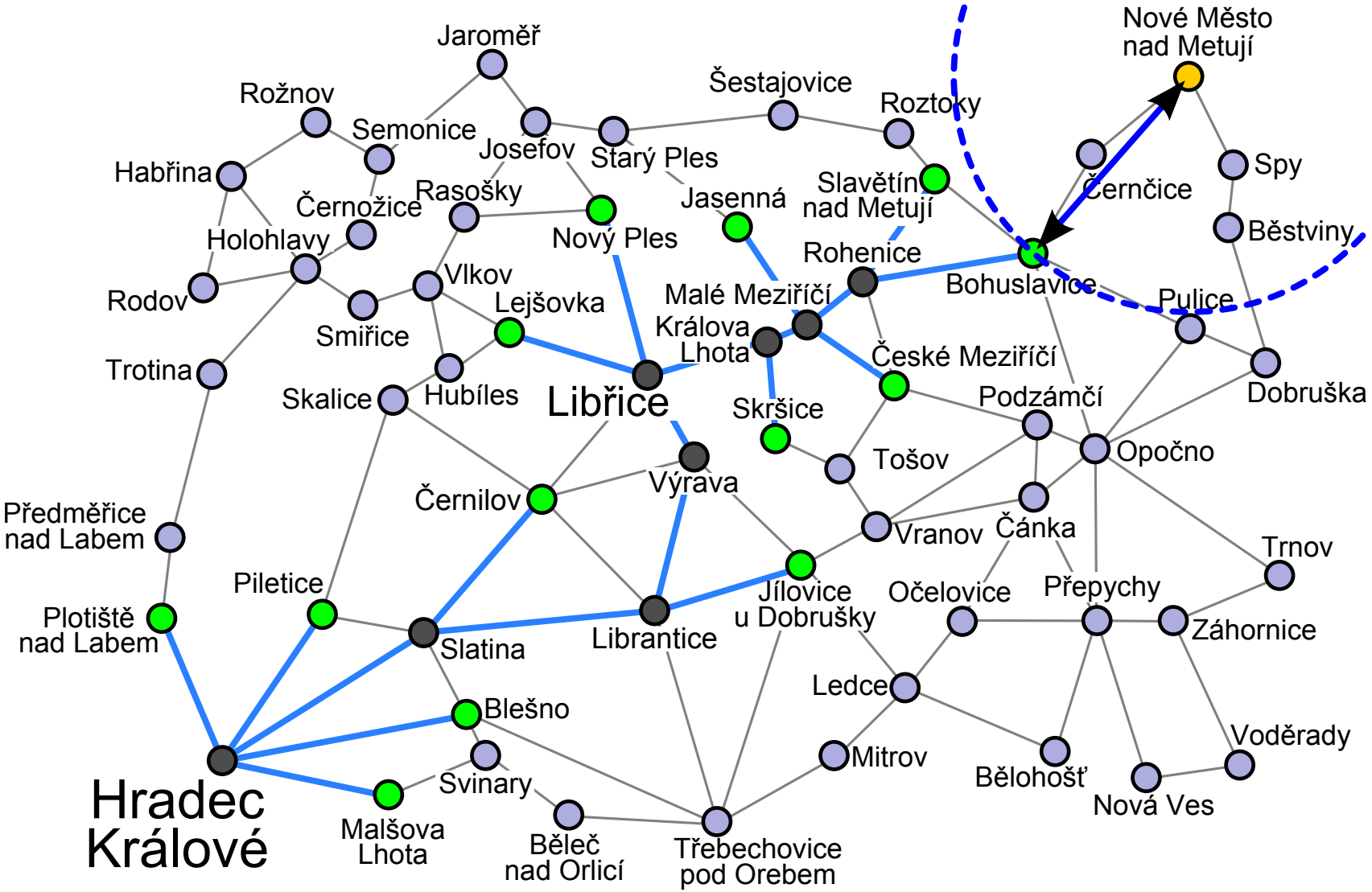
Greedy Search: Example



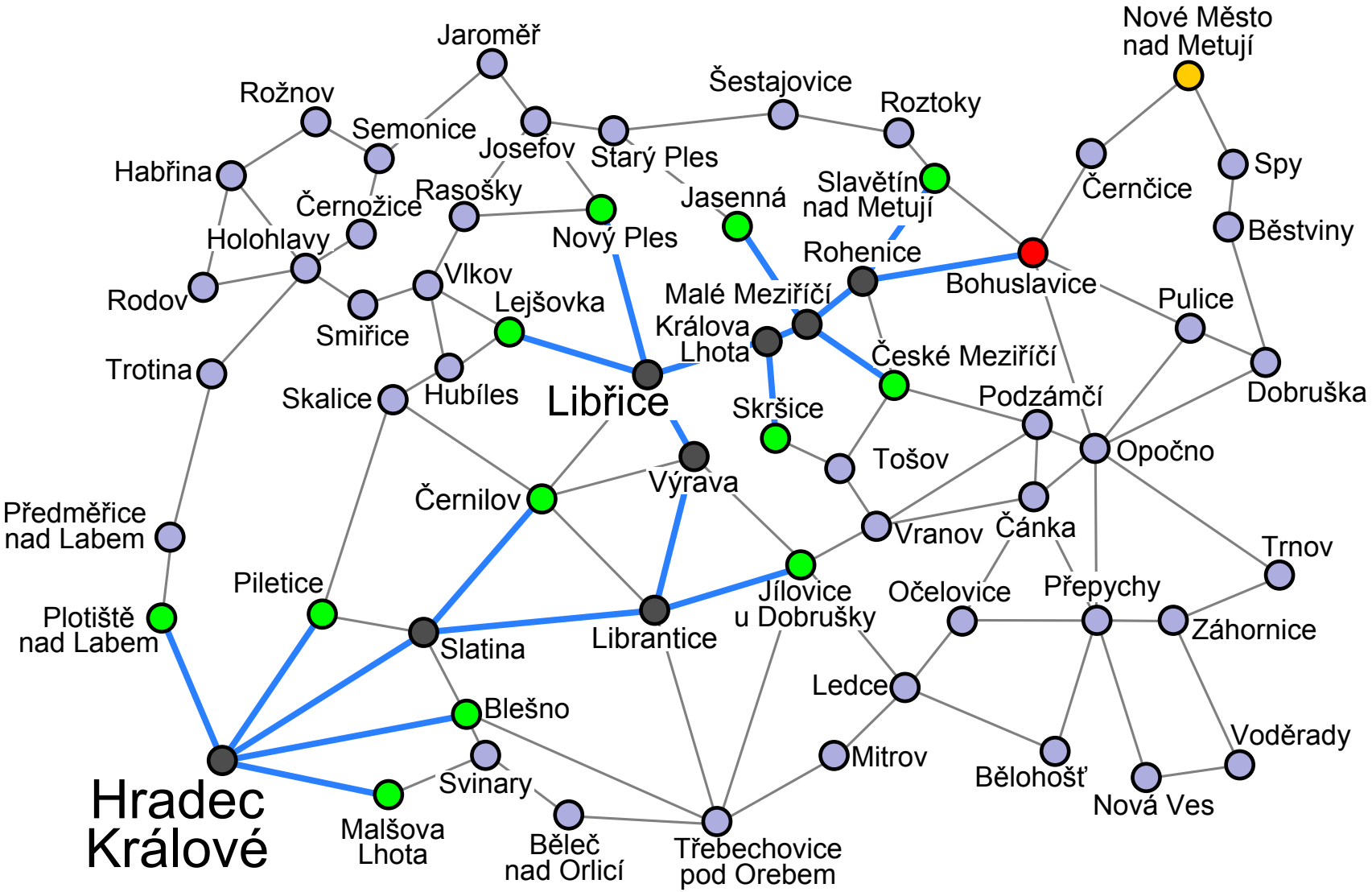
Greedy Search: Example



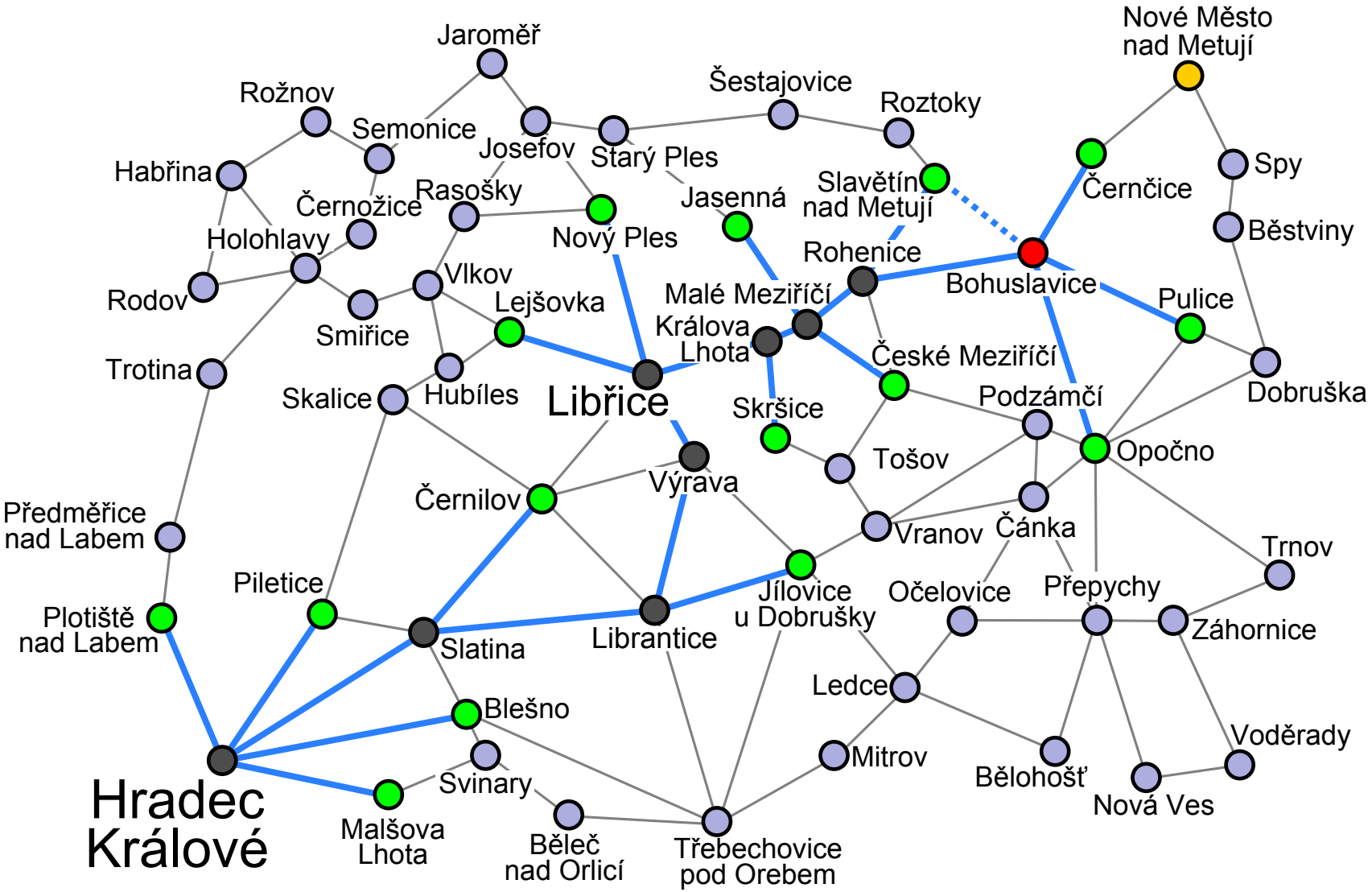
Greedy Search: Example



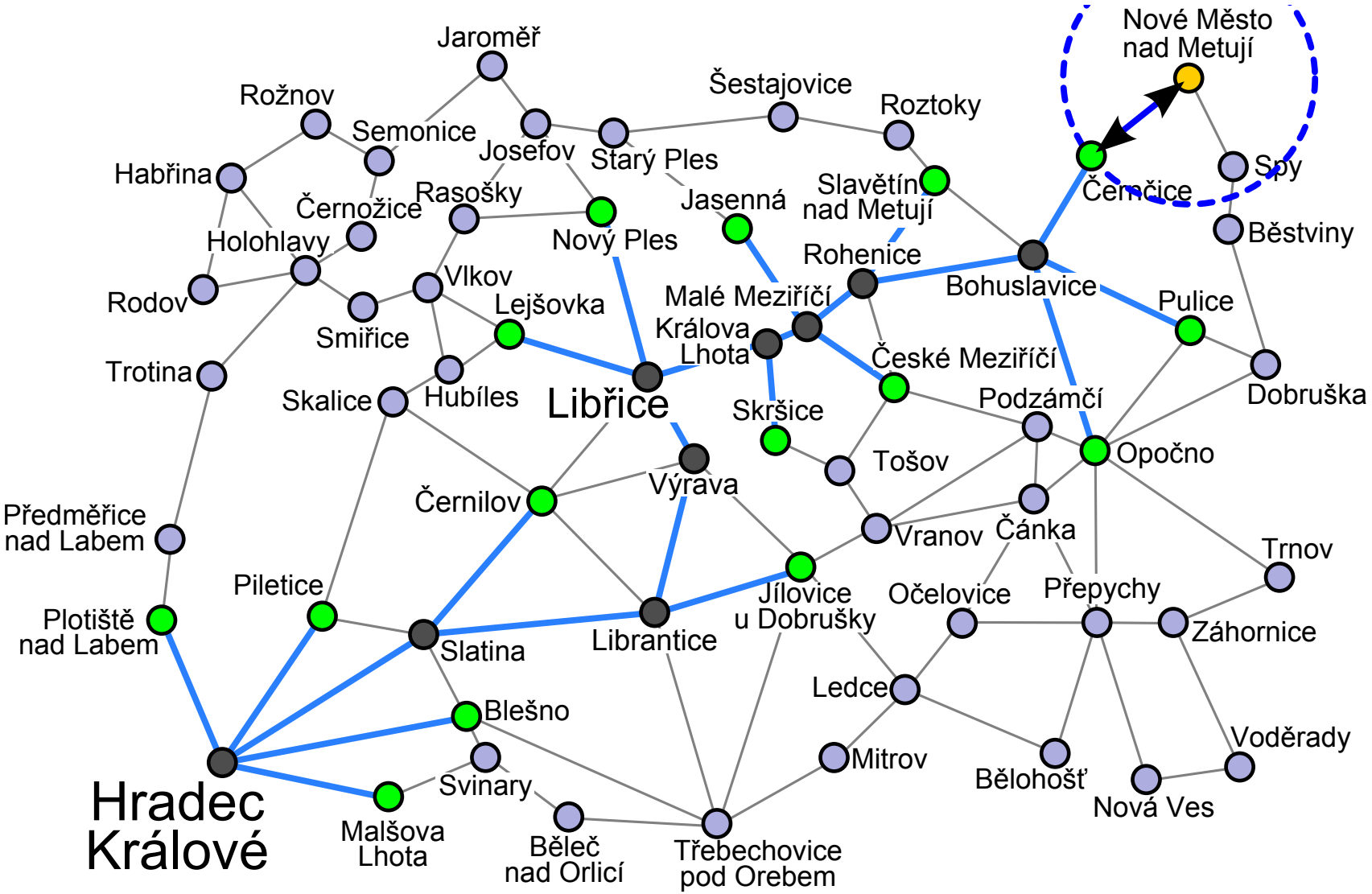
Greedy Search: Example



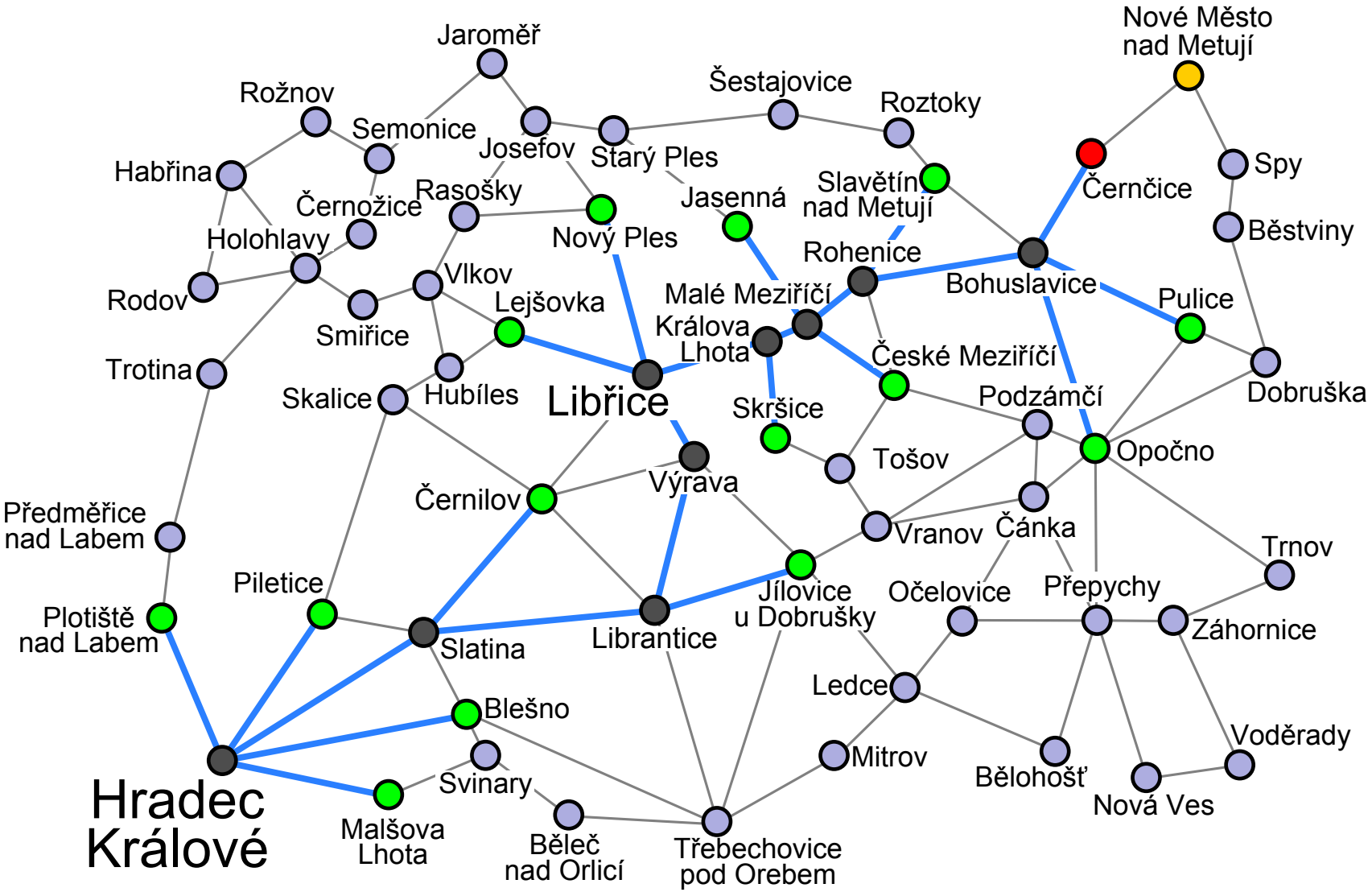
Greedy Search: Example



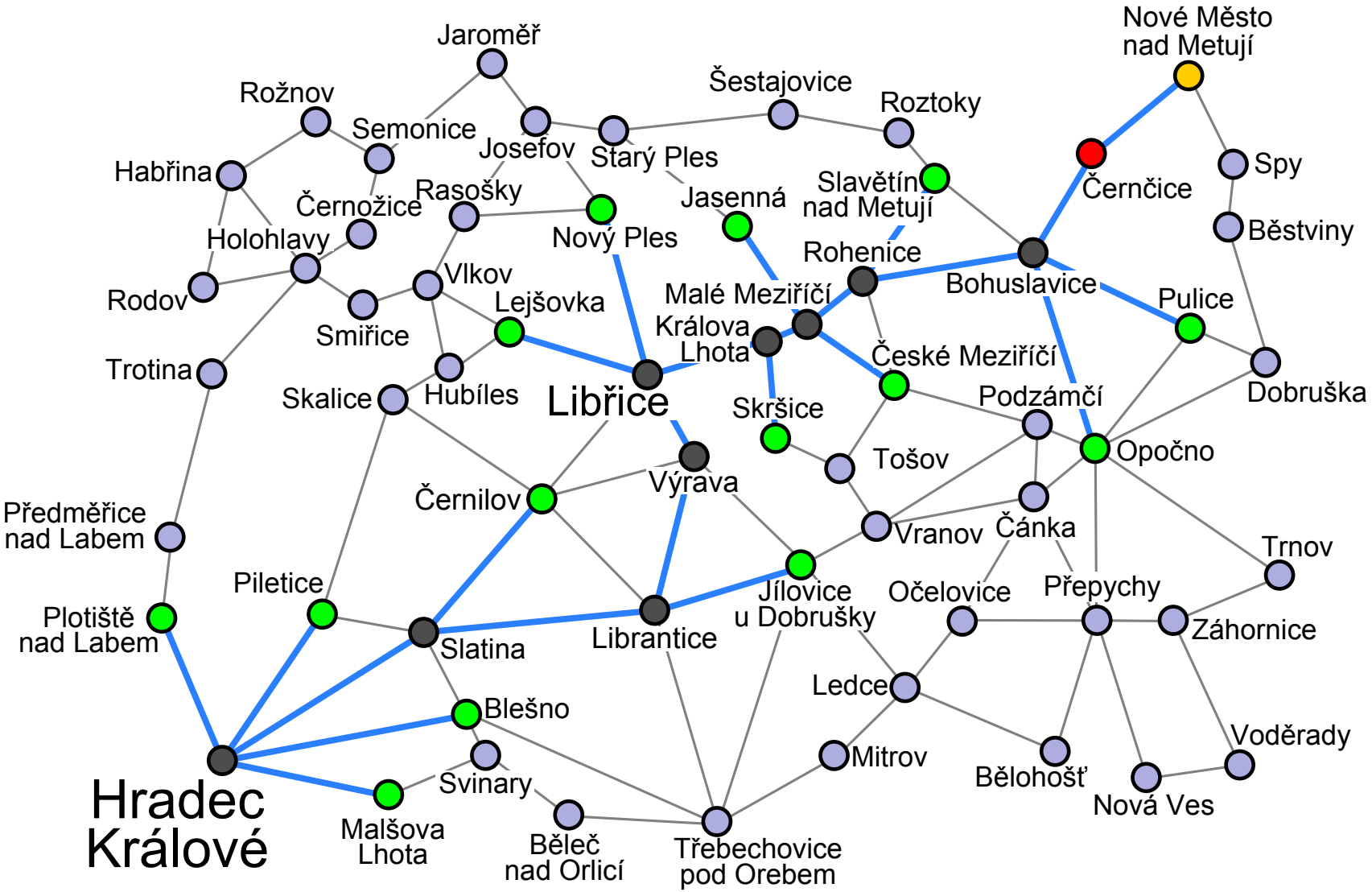
Greedy Search: Example



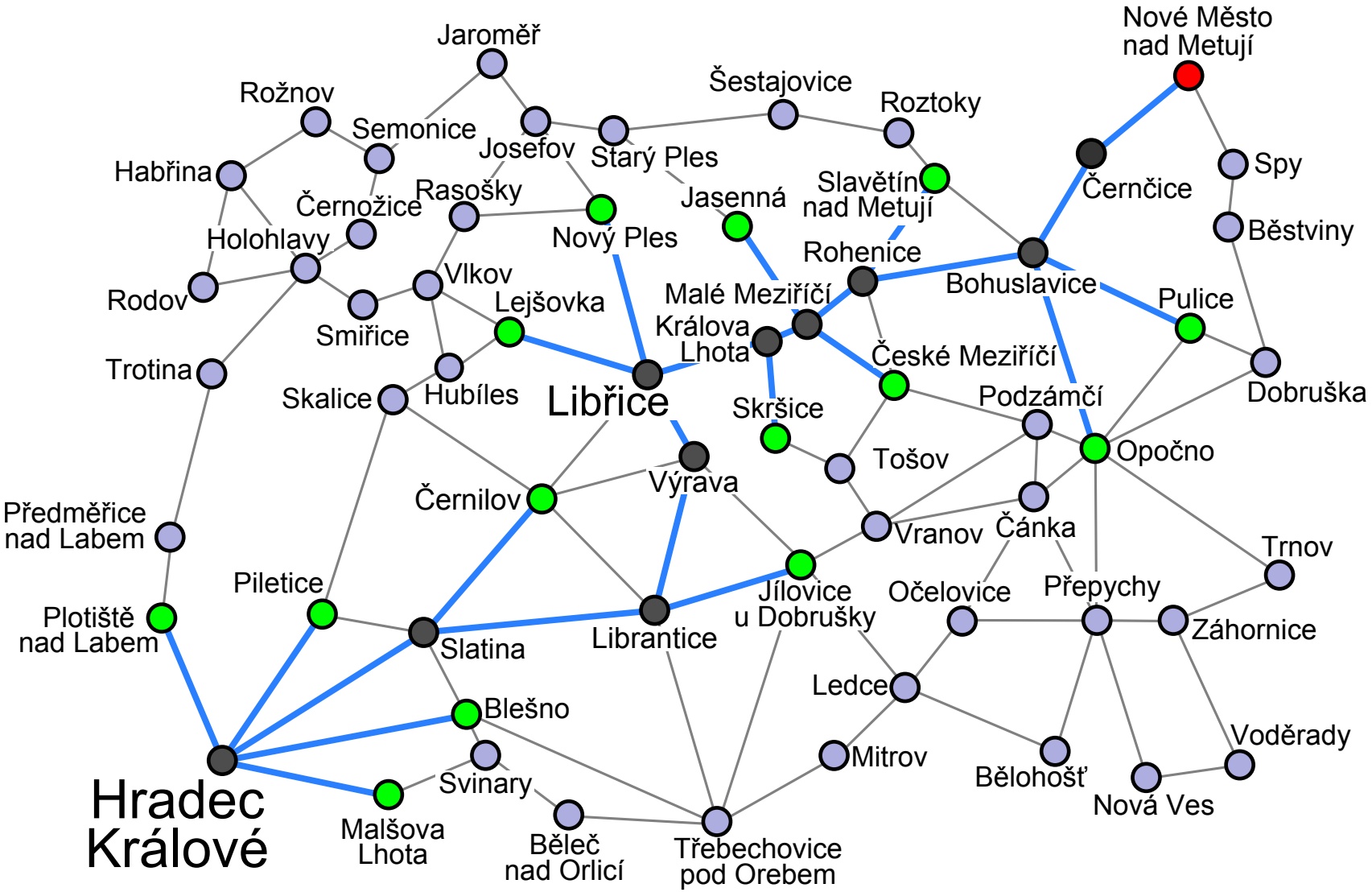
Greedy Search: Example



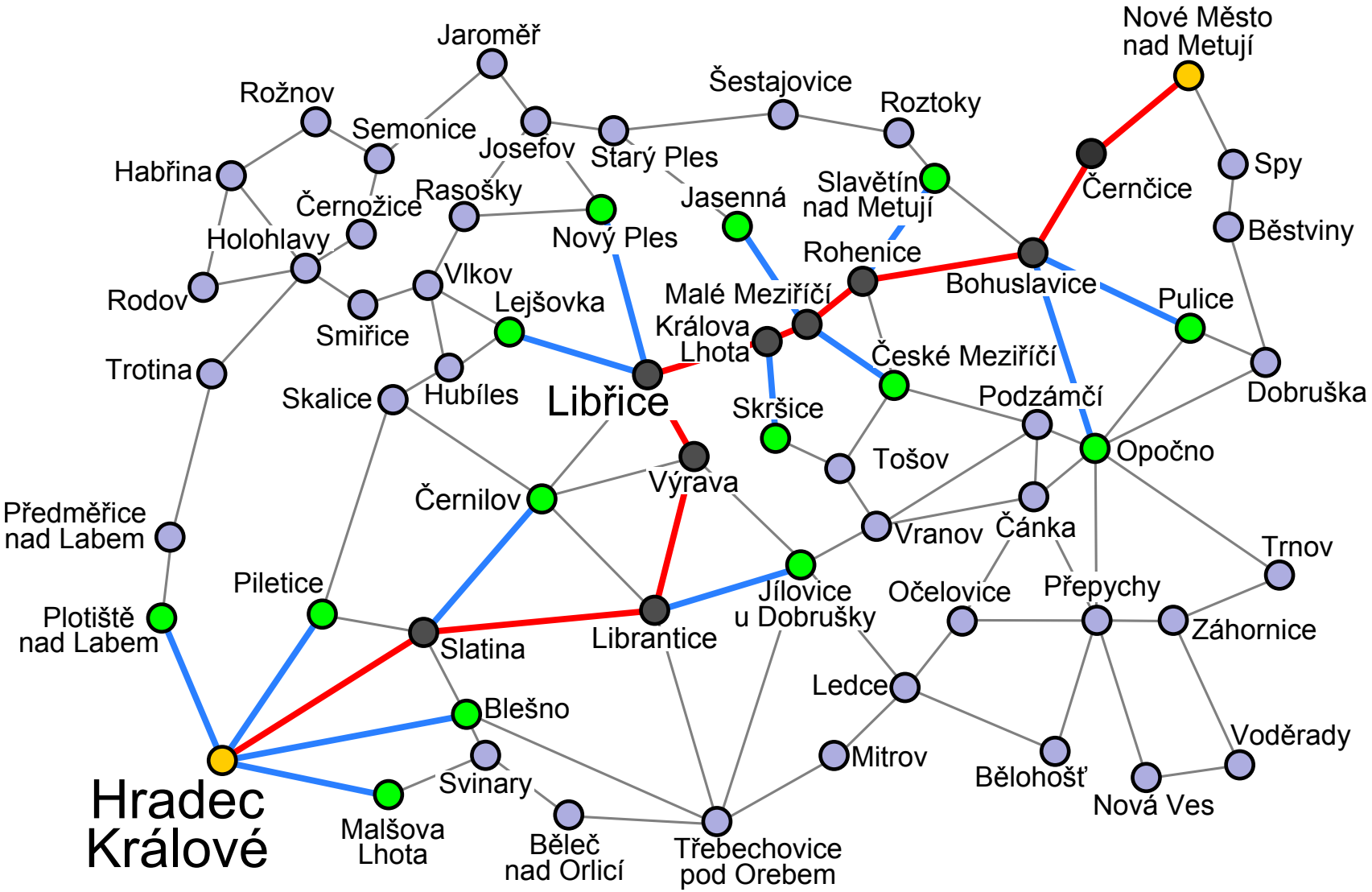
Greedy Search: Example



Greedy Search: Example



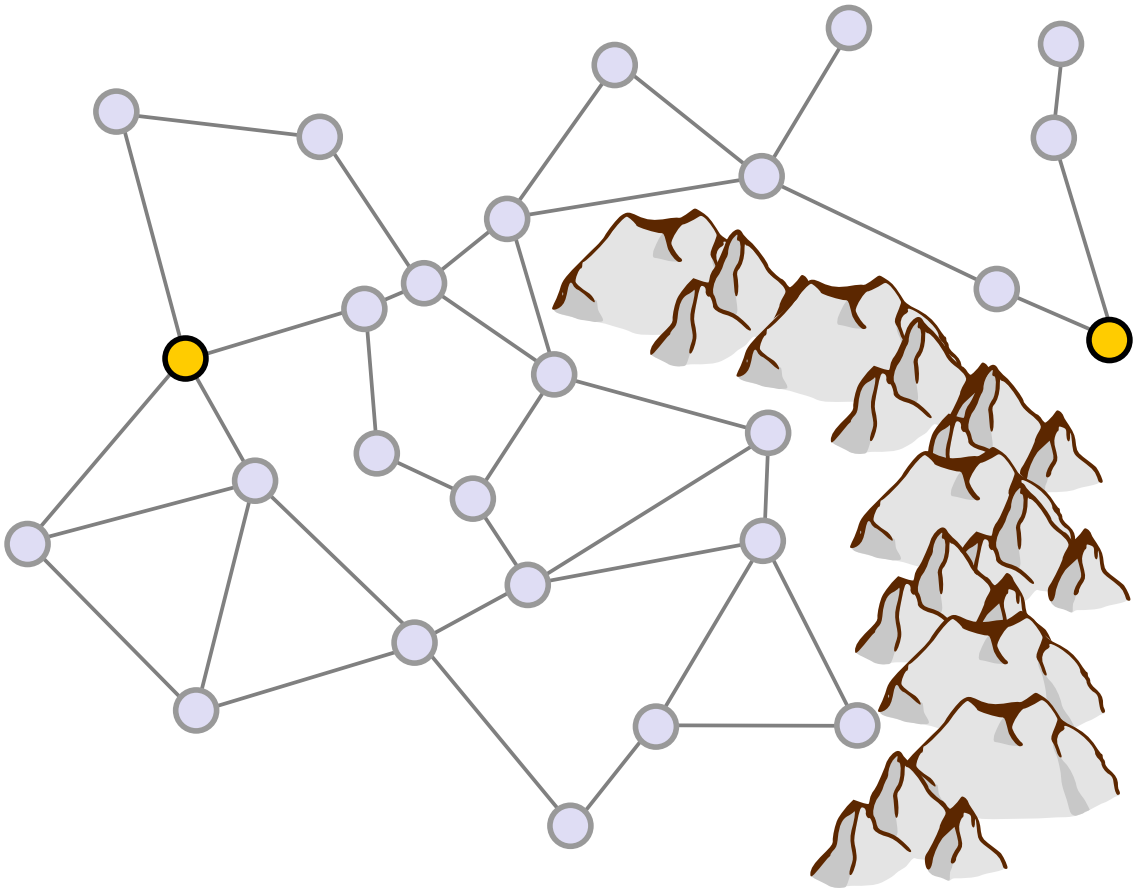
Greedy Search: Example



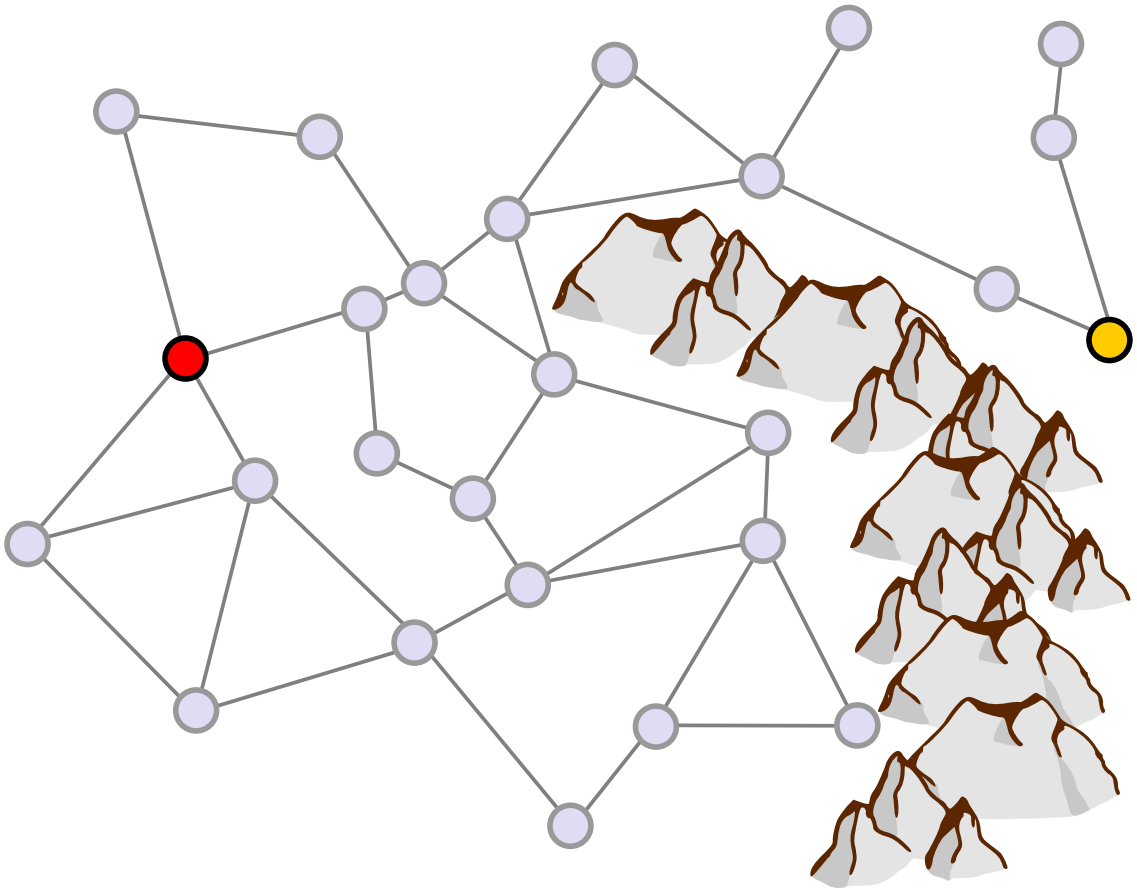
Algorithm 2 Greedy search

```
 1: open ← init_priority_queue(); closed ← {}
 2: prev ← init_table()
 3: for all s ∈ I do
 4:     enqueue(open, s, h(s))
 5: end for
 6: while ¬empty(open) do
 7:     x ← dequeue(open)
 8:     if x ∈ G then
 9:         return reconstruct_path(prev, x)
10:     end if
11:     for all y ∈ neighbors(x) do
12:         if y ∉ (open ∪ closed) then
13:             enqueue(open, y, h(y))
14:             prev[y] ← x
15:         end if
16:     end for
17:     closed ← closed ∪ {x}
18: end while
```

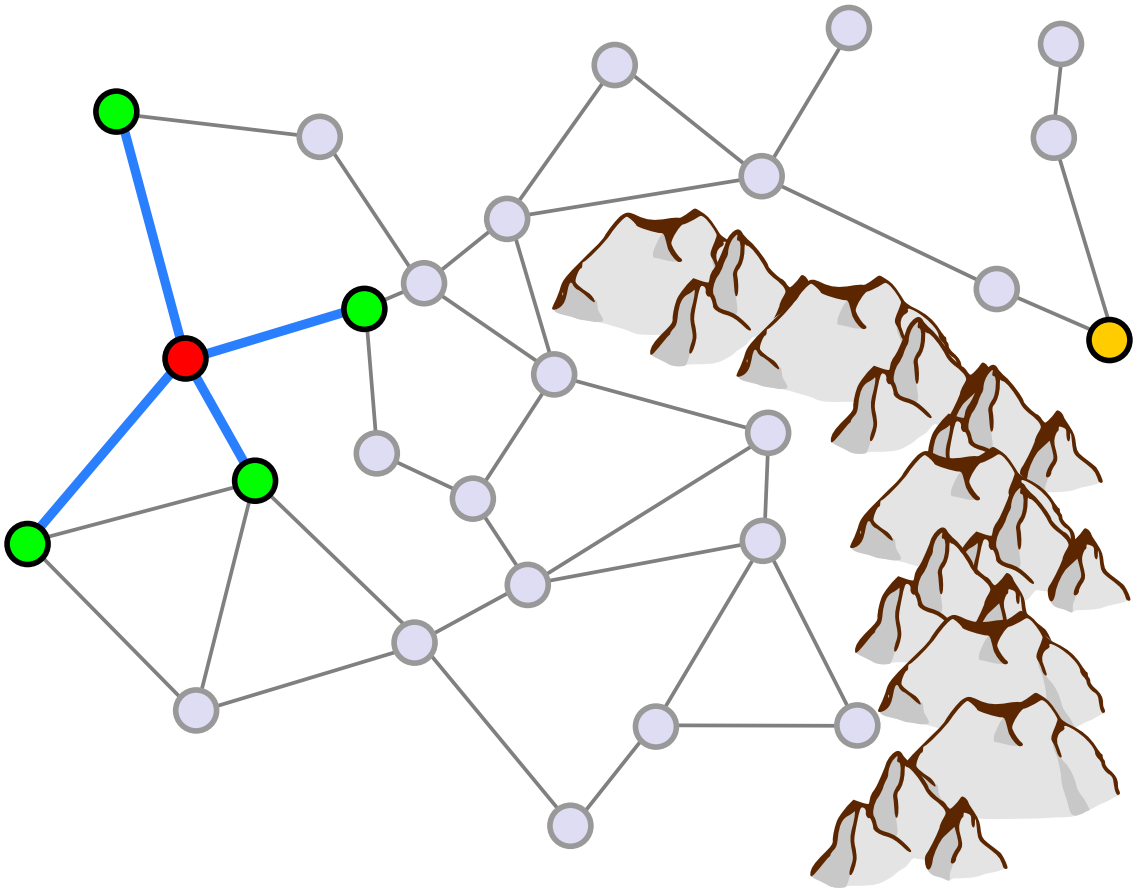
Greedy Search: Air Distance Failure



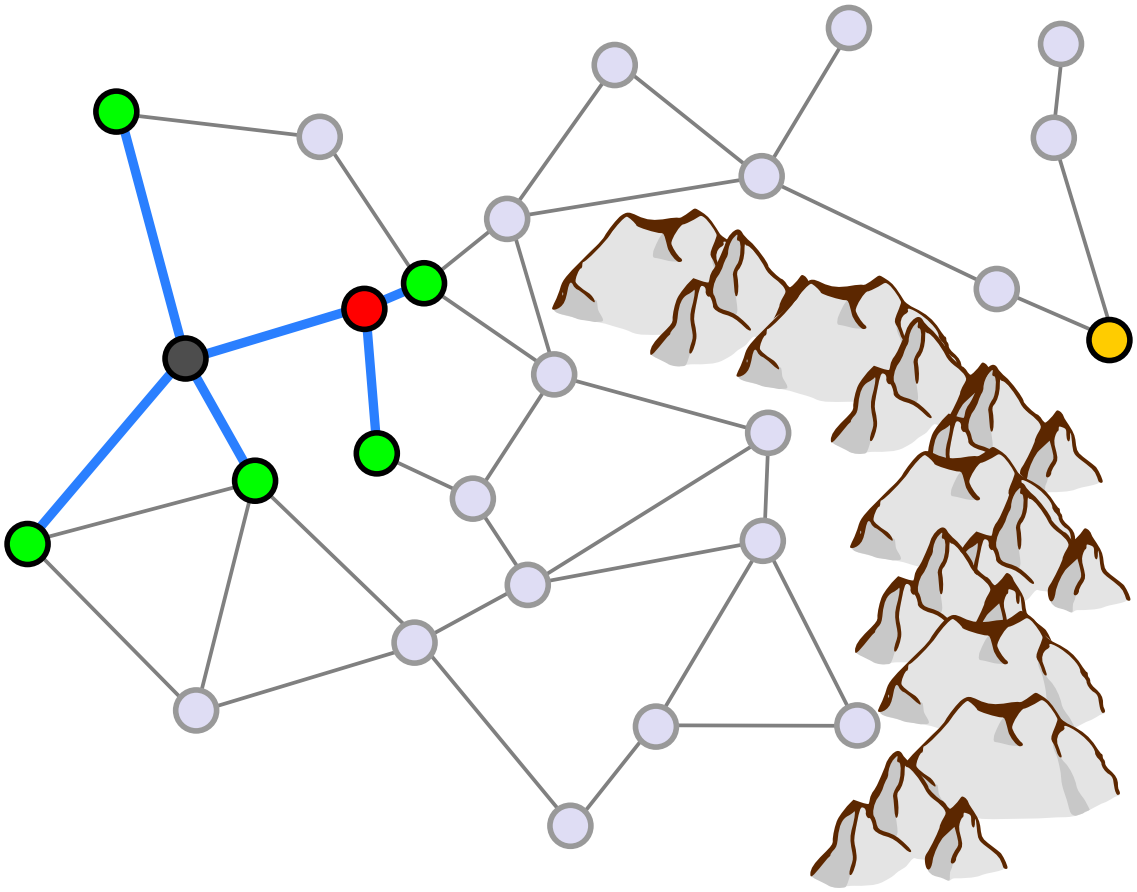
Greedy Search: Air Distance Failure



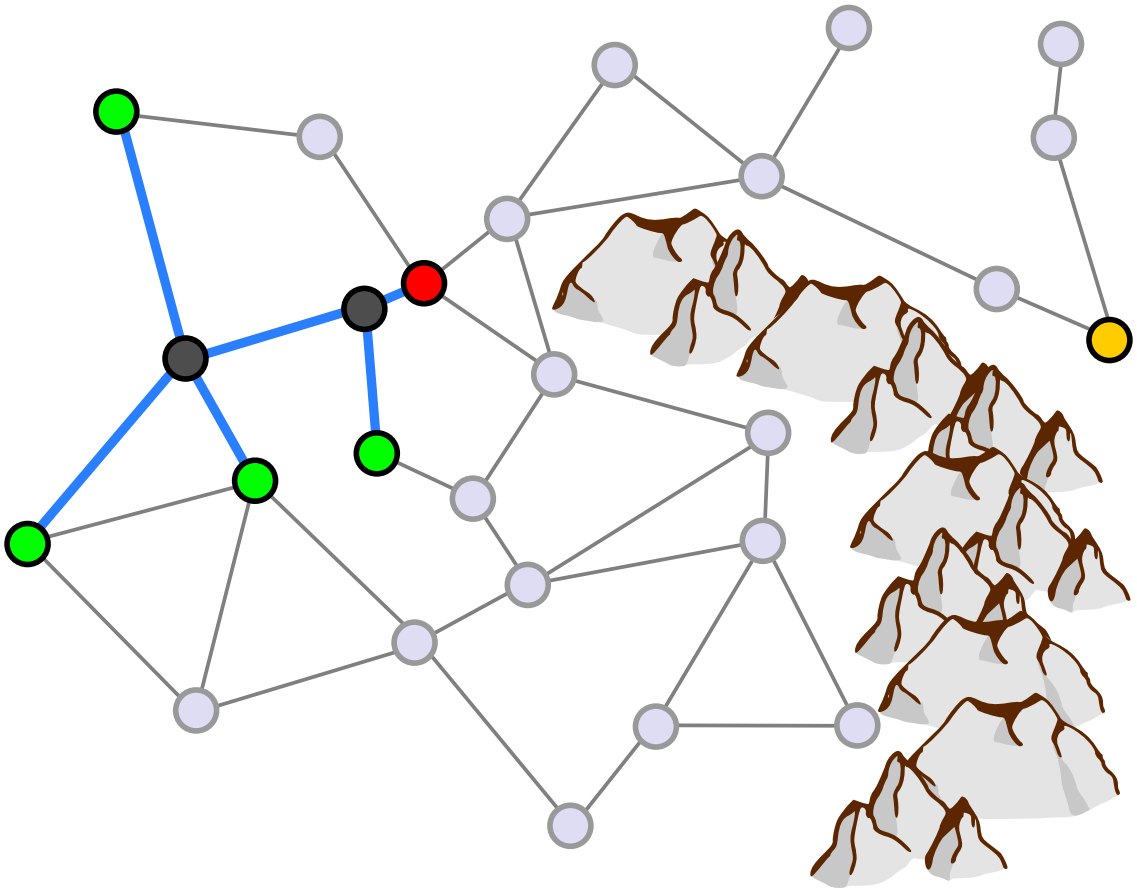
Greedy Search: Air Distance Failure



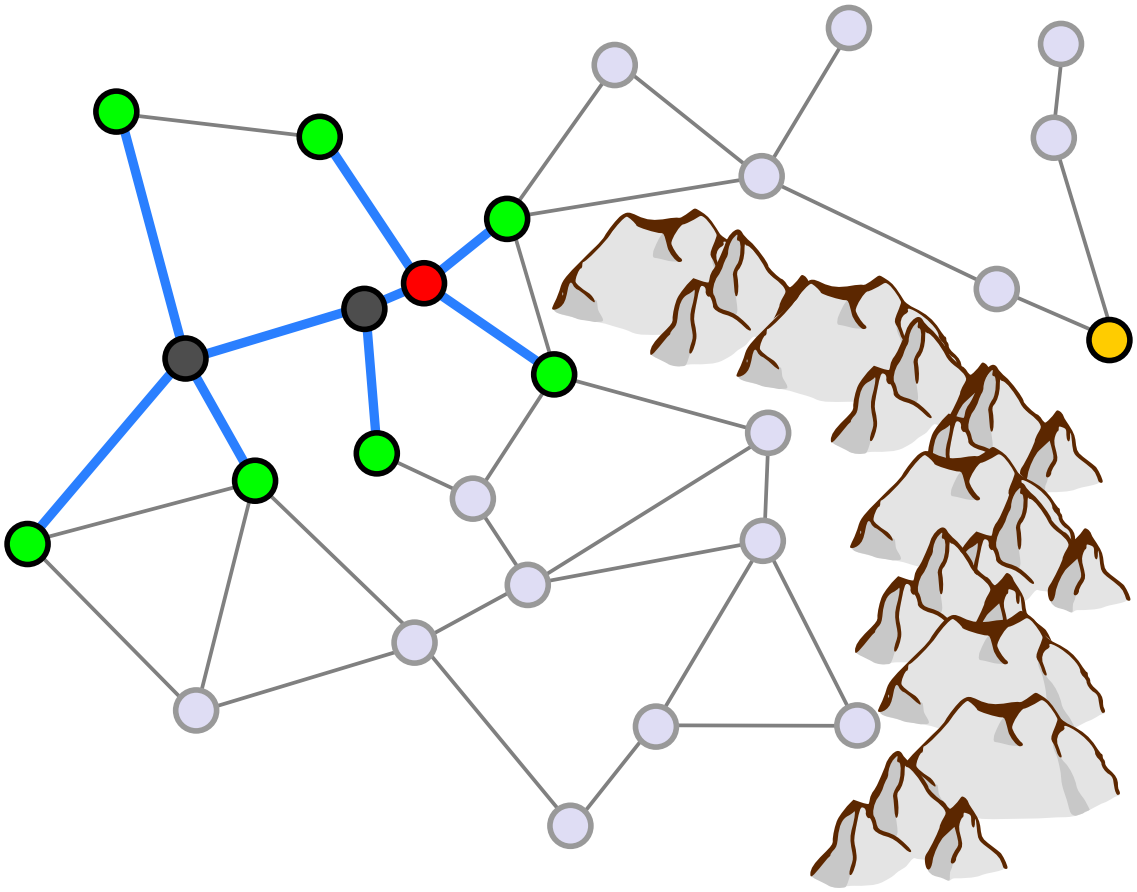
Greedy Search: Air Distance Failure



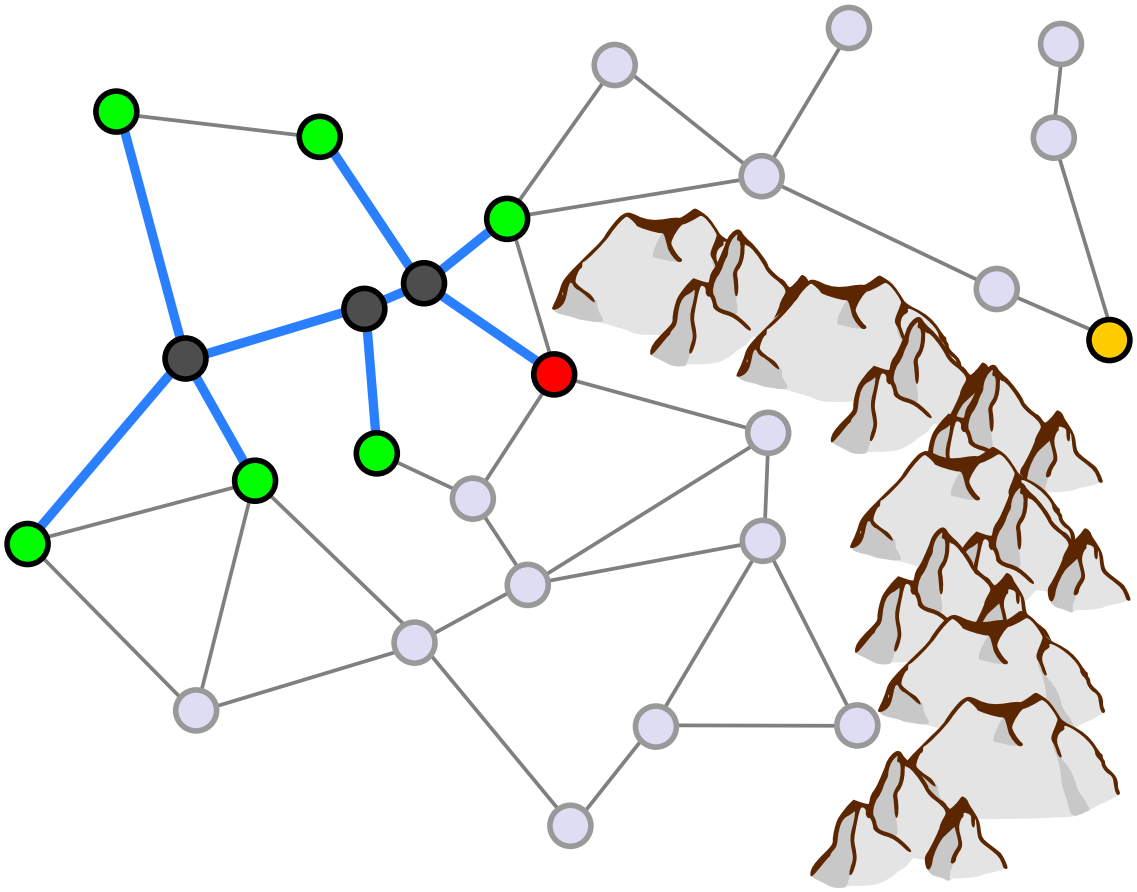
Greedy Search: Air Distance Failure



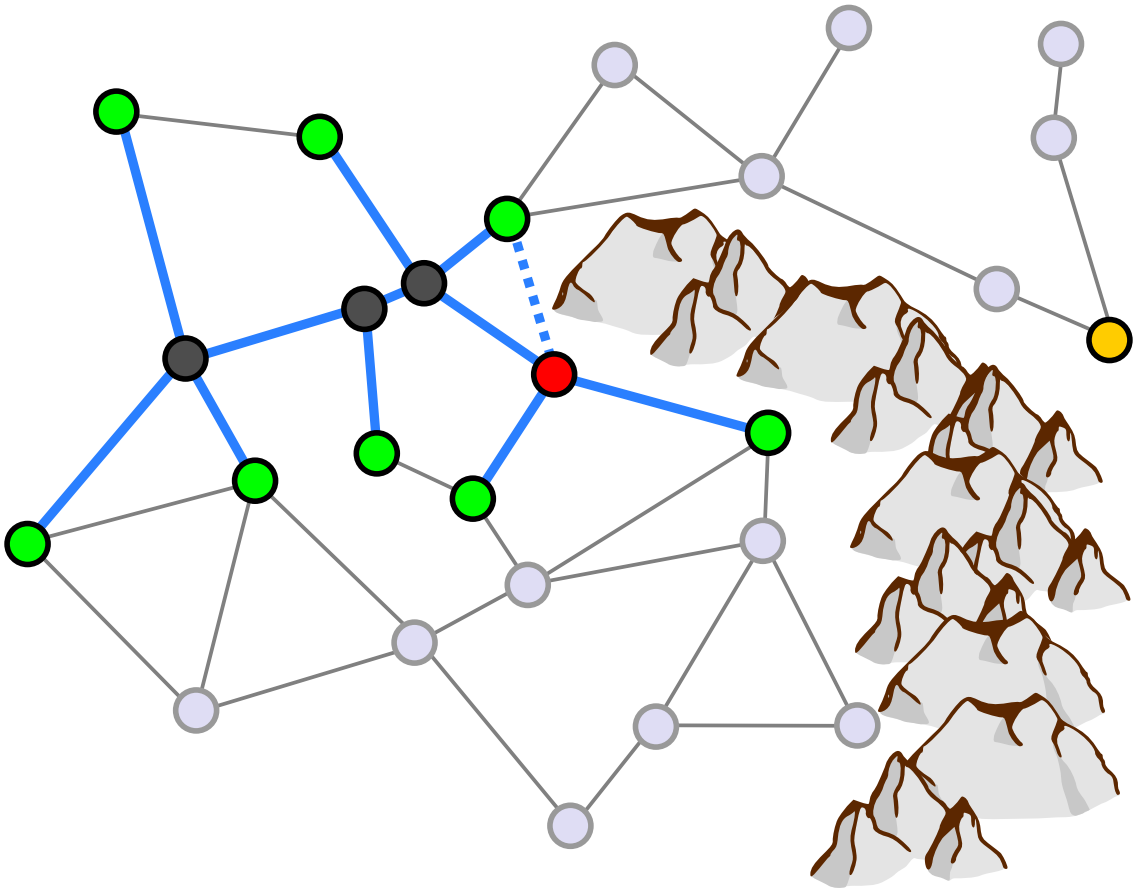
Greedy Search: Air Distance Failure



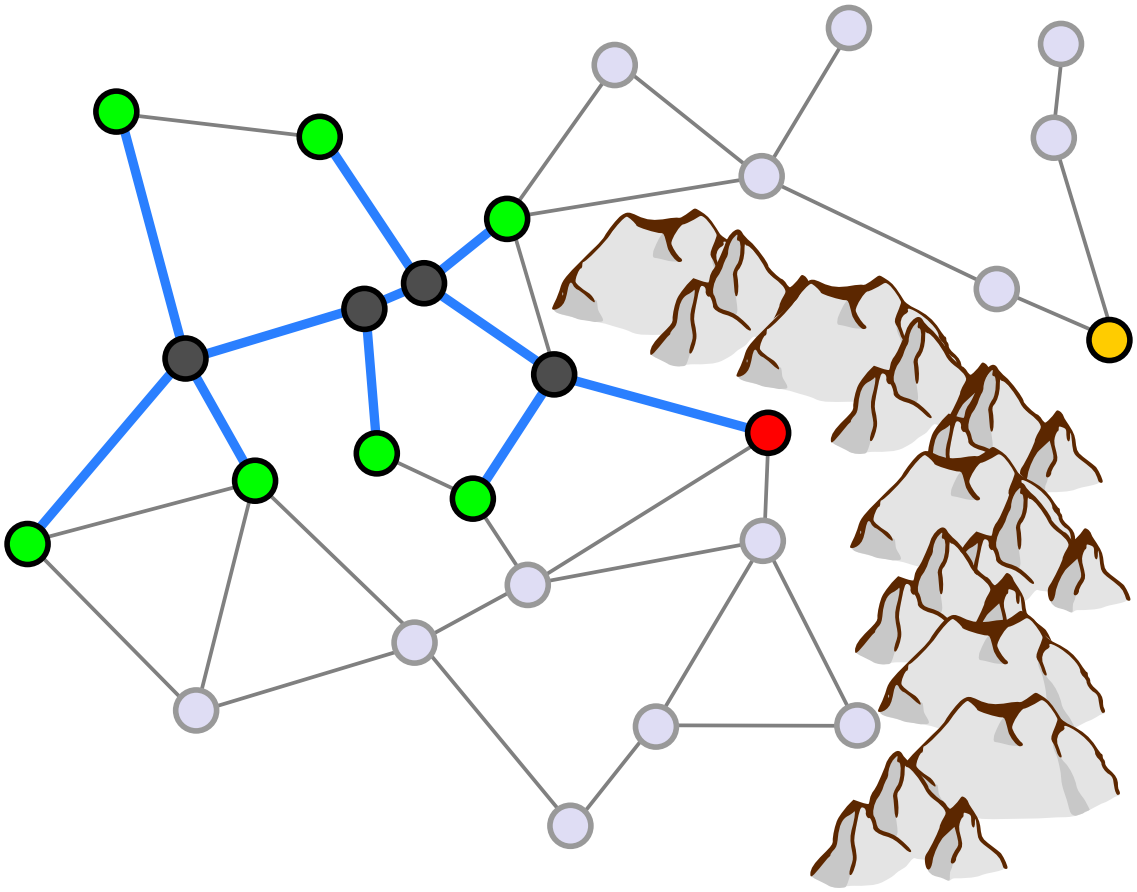
Greedy Search: Air Distance Failure



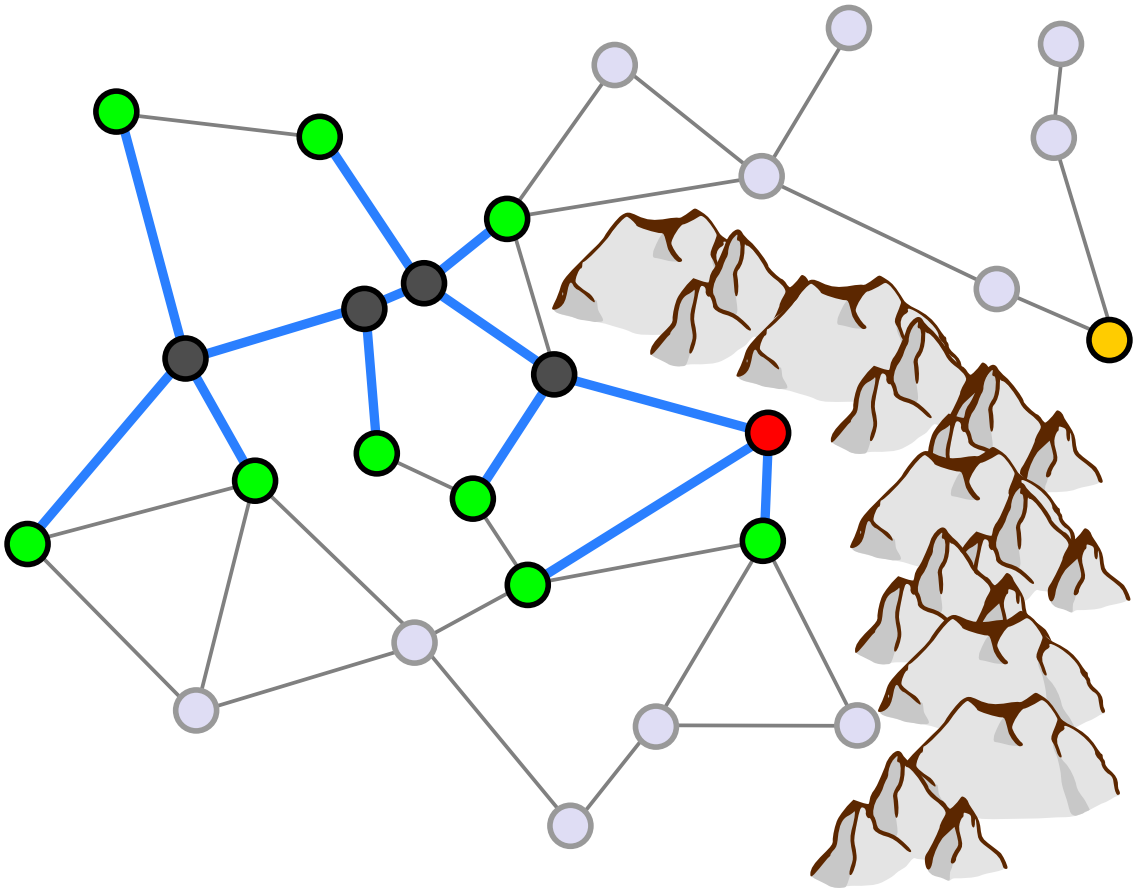
Greedy Search: Air Distance Failure



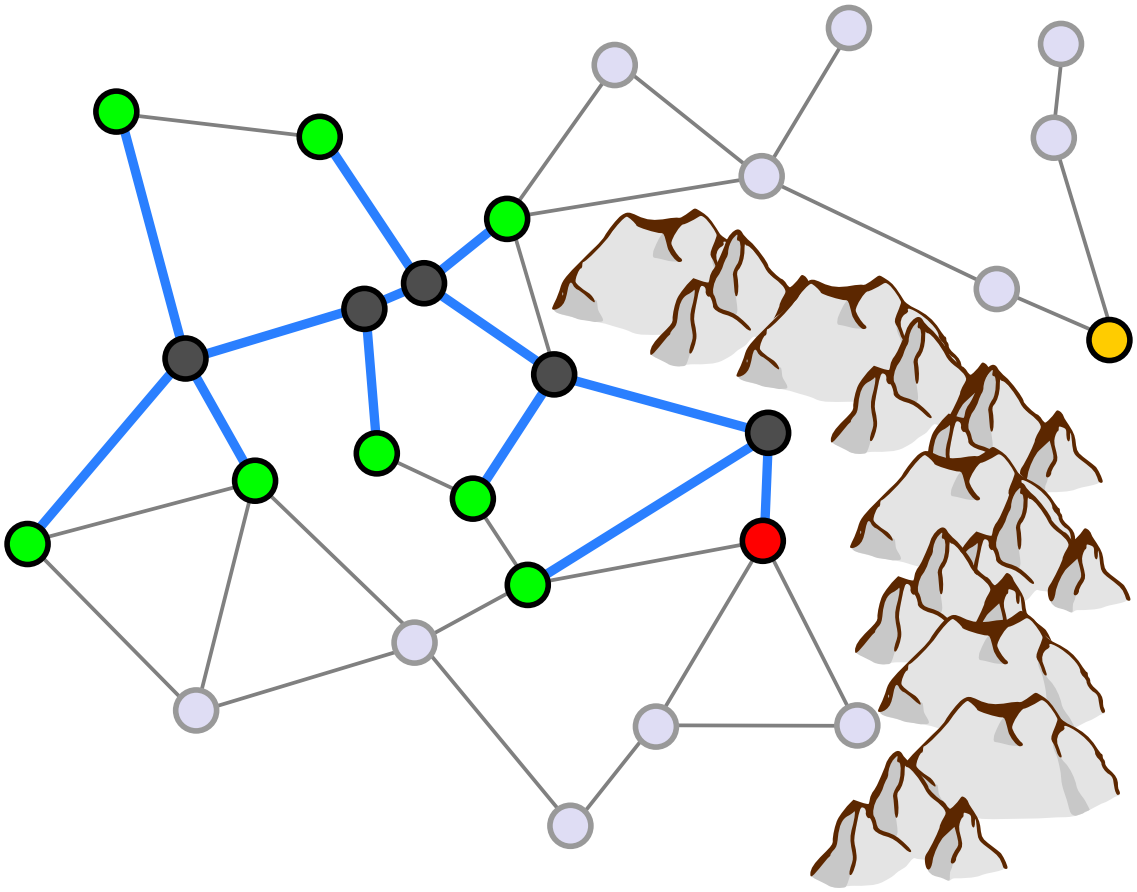
Greedy Search: Air Distance Failure



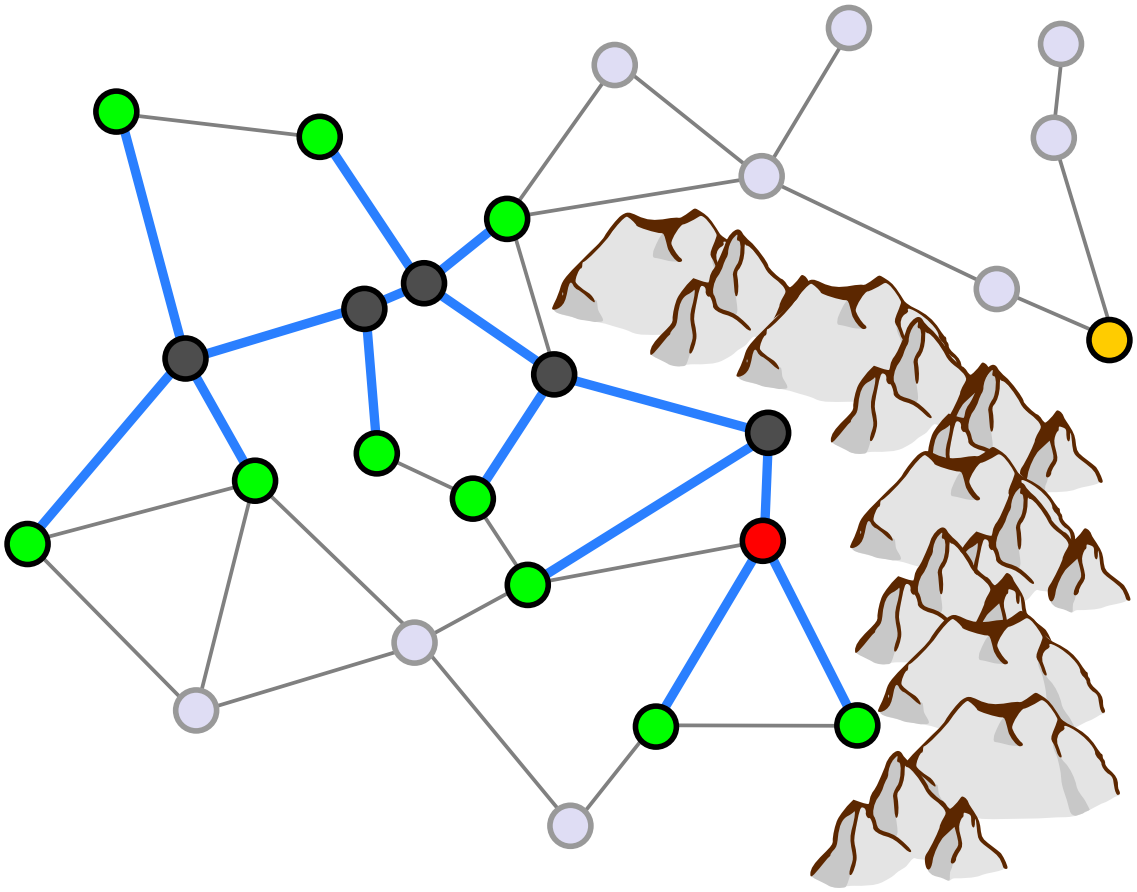
Greedy Search: Air Distance Failure



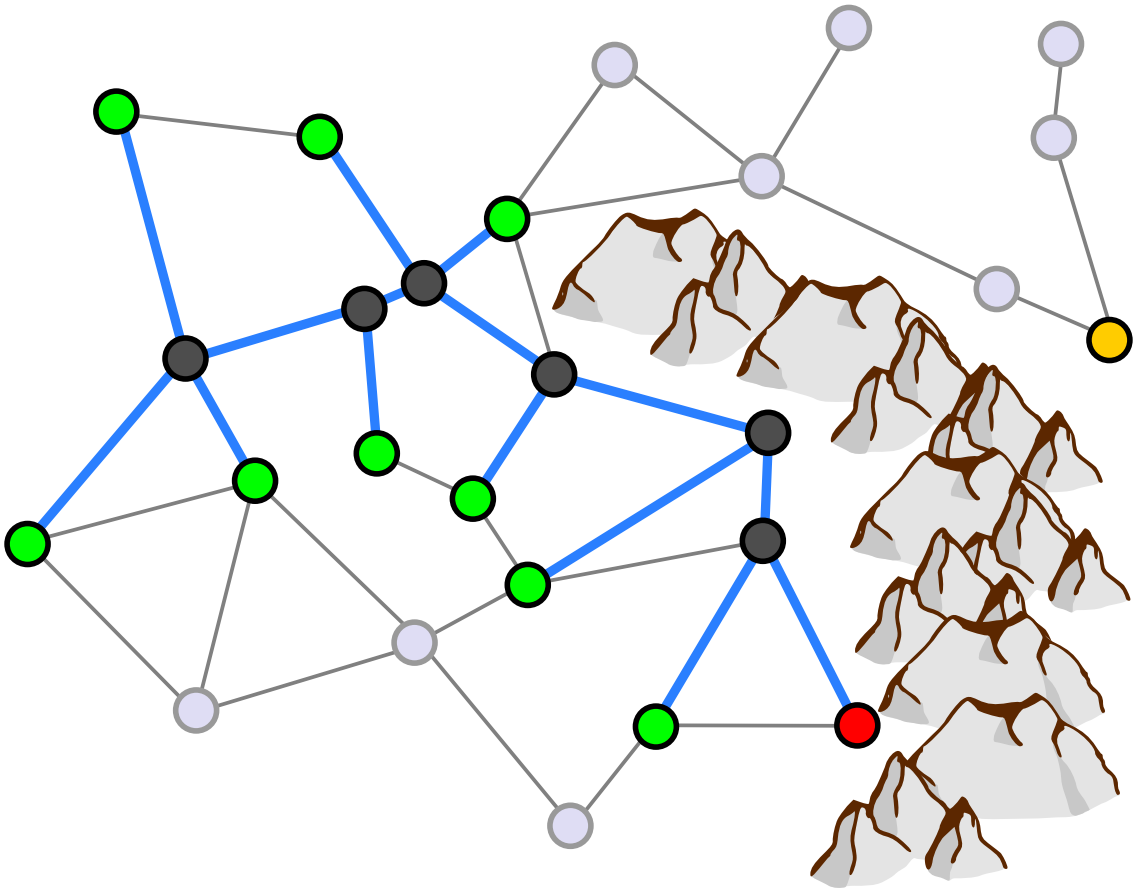
Greedy Search: Air Distance Failure



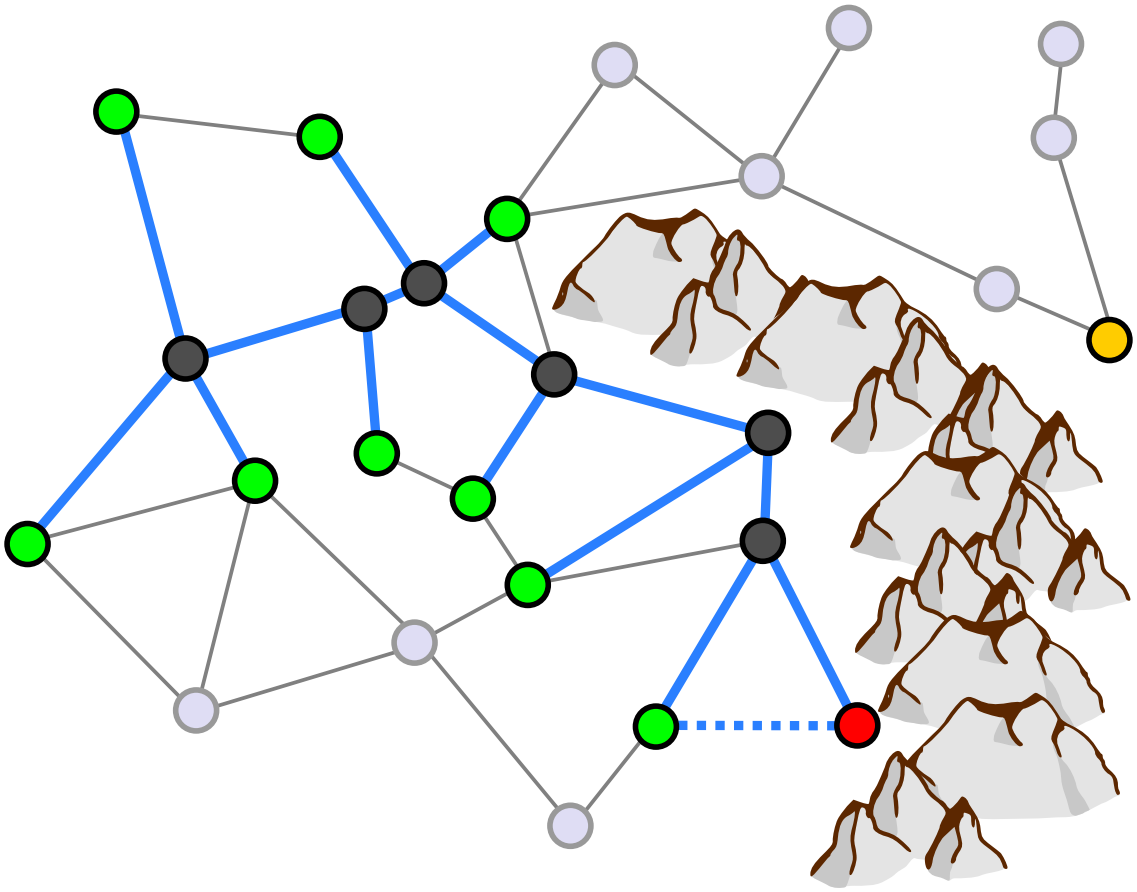
Greedy Search: Air Distance Failure



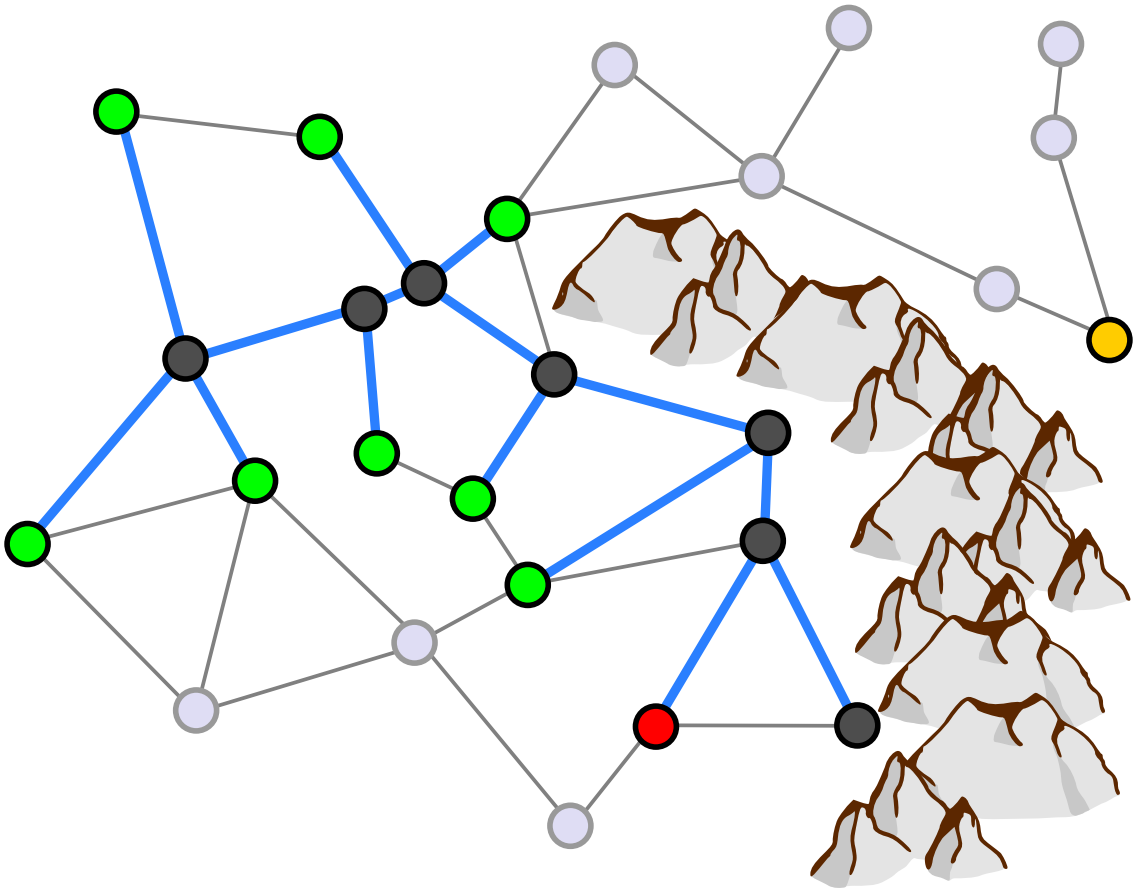
Greedy Search: Air Distance Failure



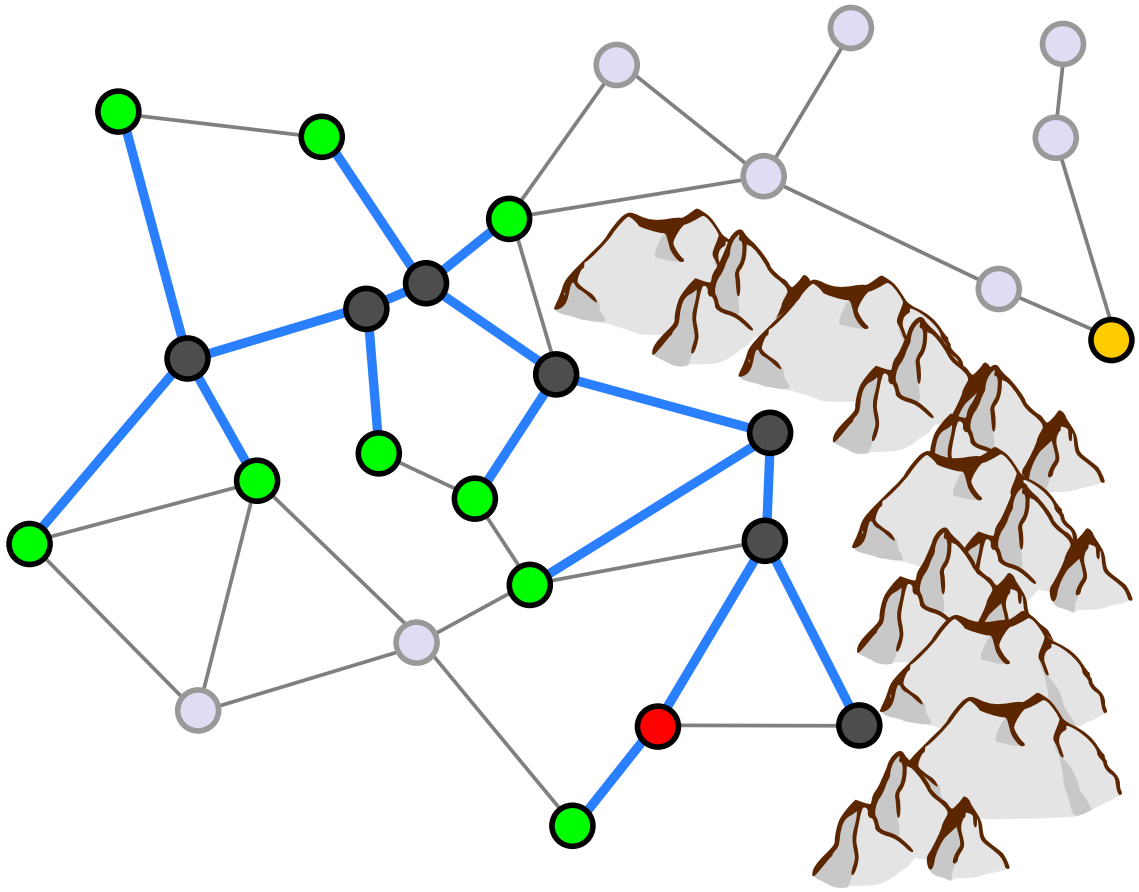
Greedy Search: Air Distance Failure



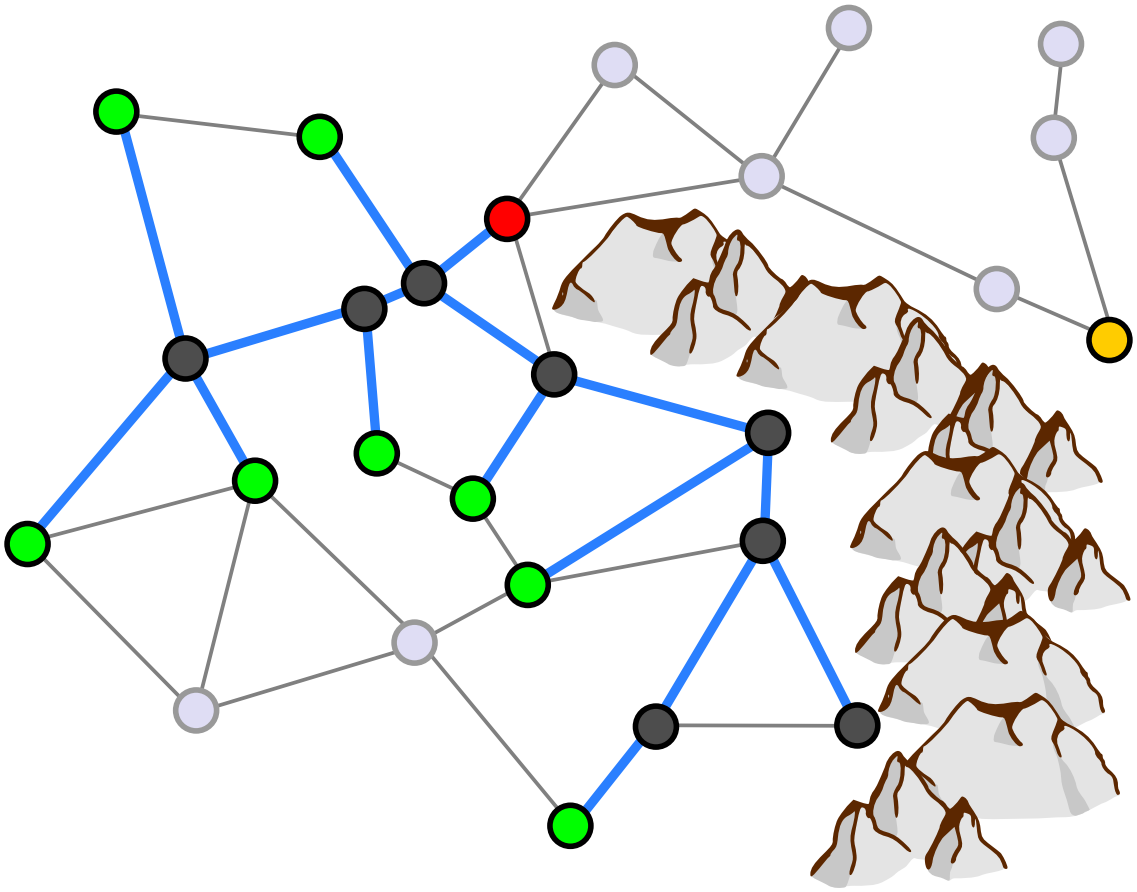
Greedy Search: Air Distance Failure



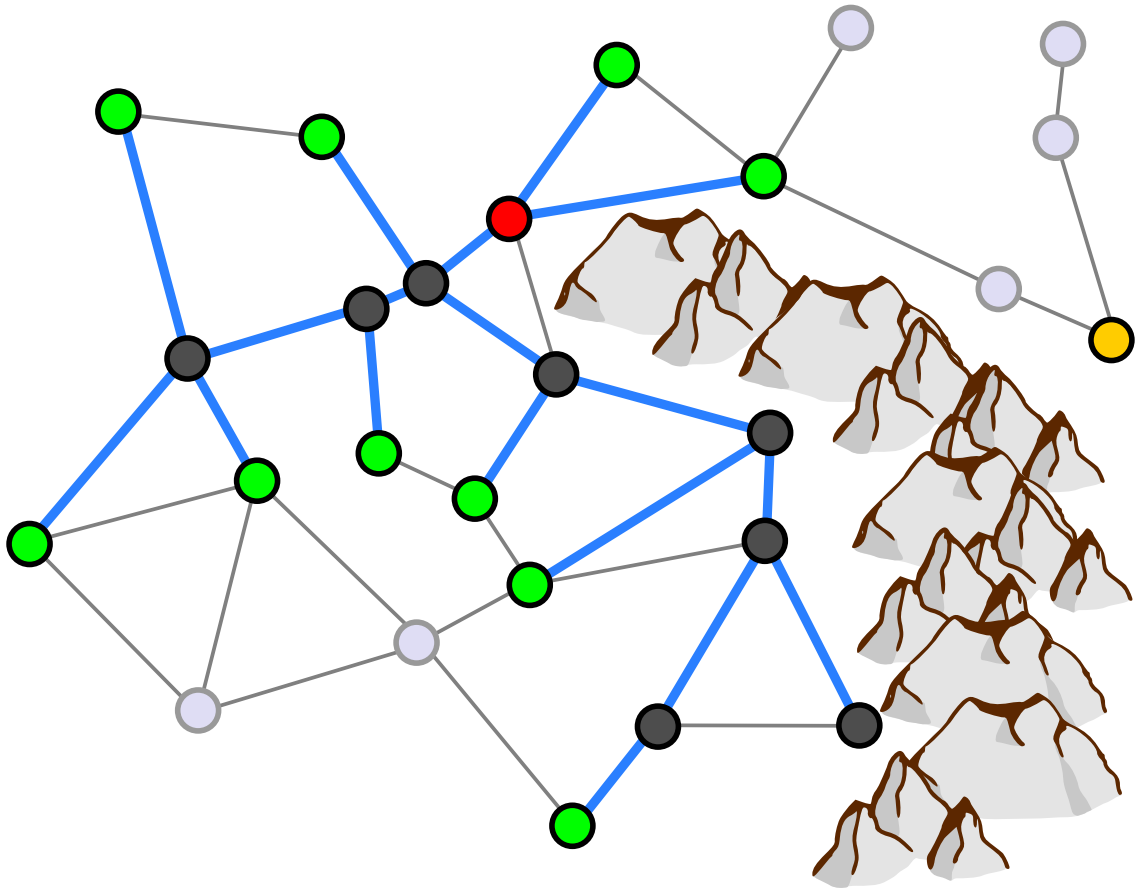
Greedy Search: Air Distance Failure



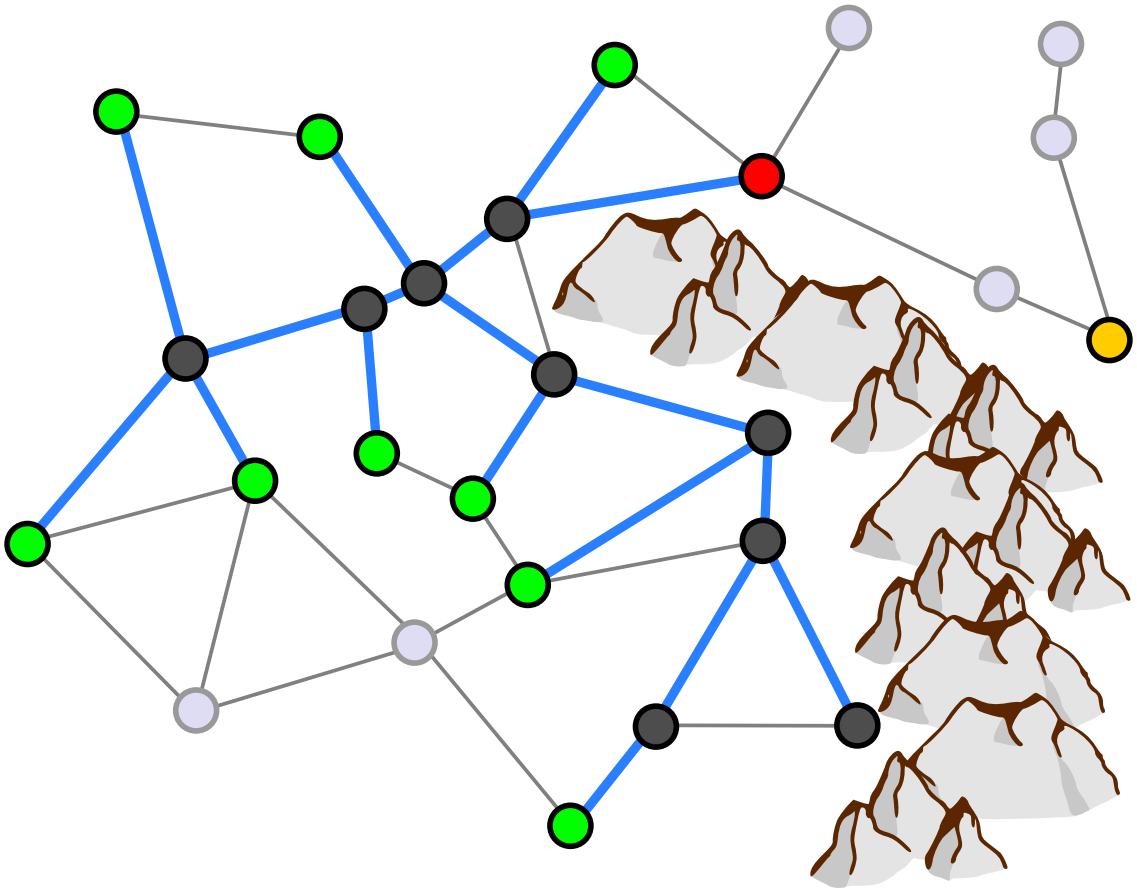
Greedy Search: Air Distance Failure



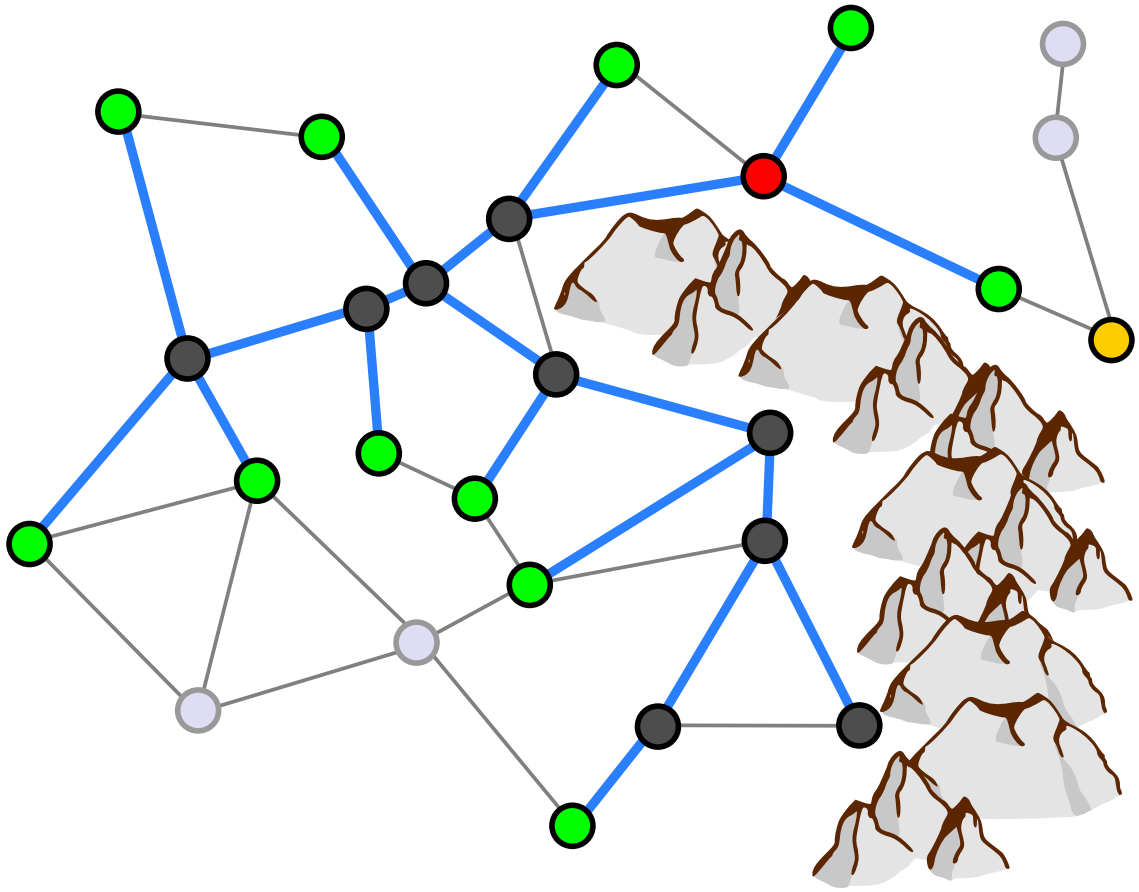
Greedy Search: Air Distance Failure



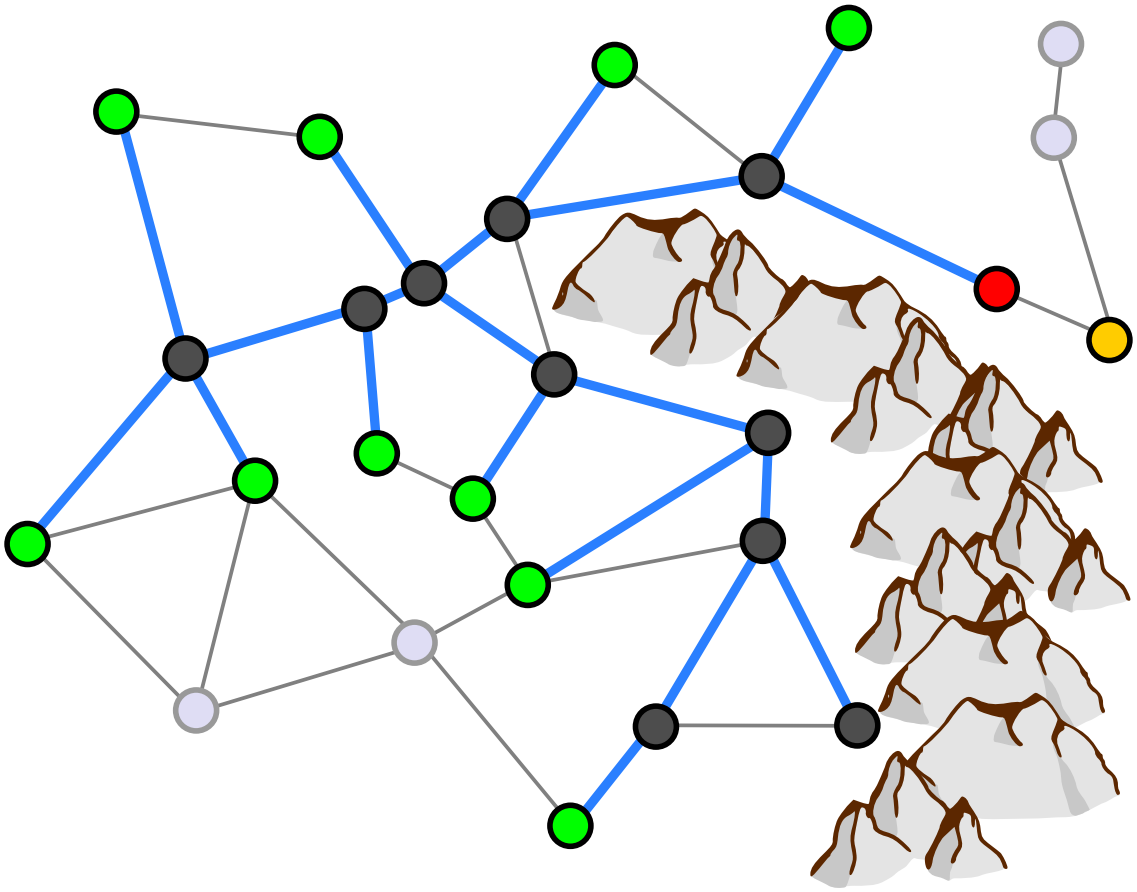
Greedy Search: Air Distance Failure



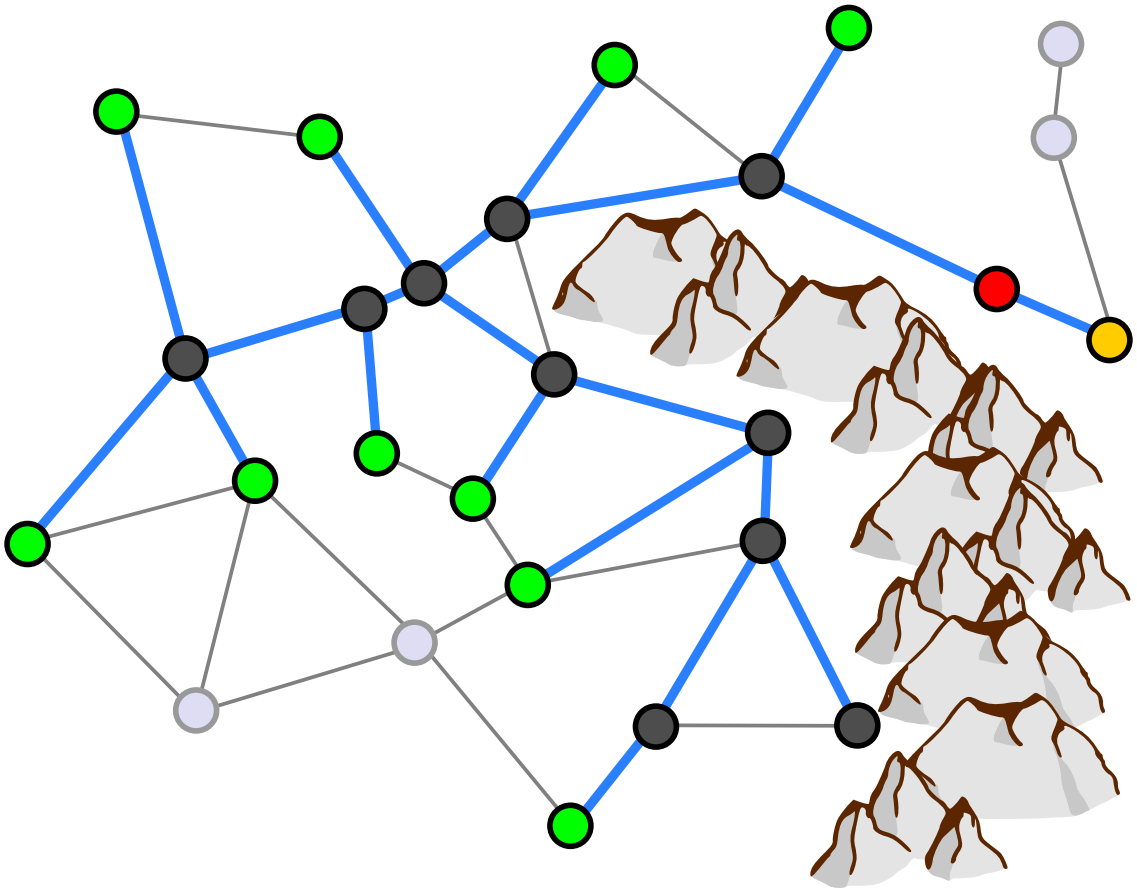
Greedy Search: Air Distance Failure



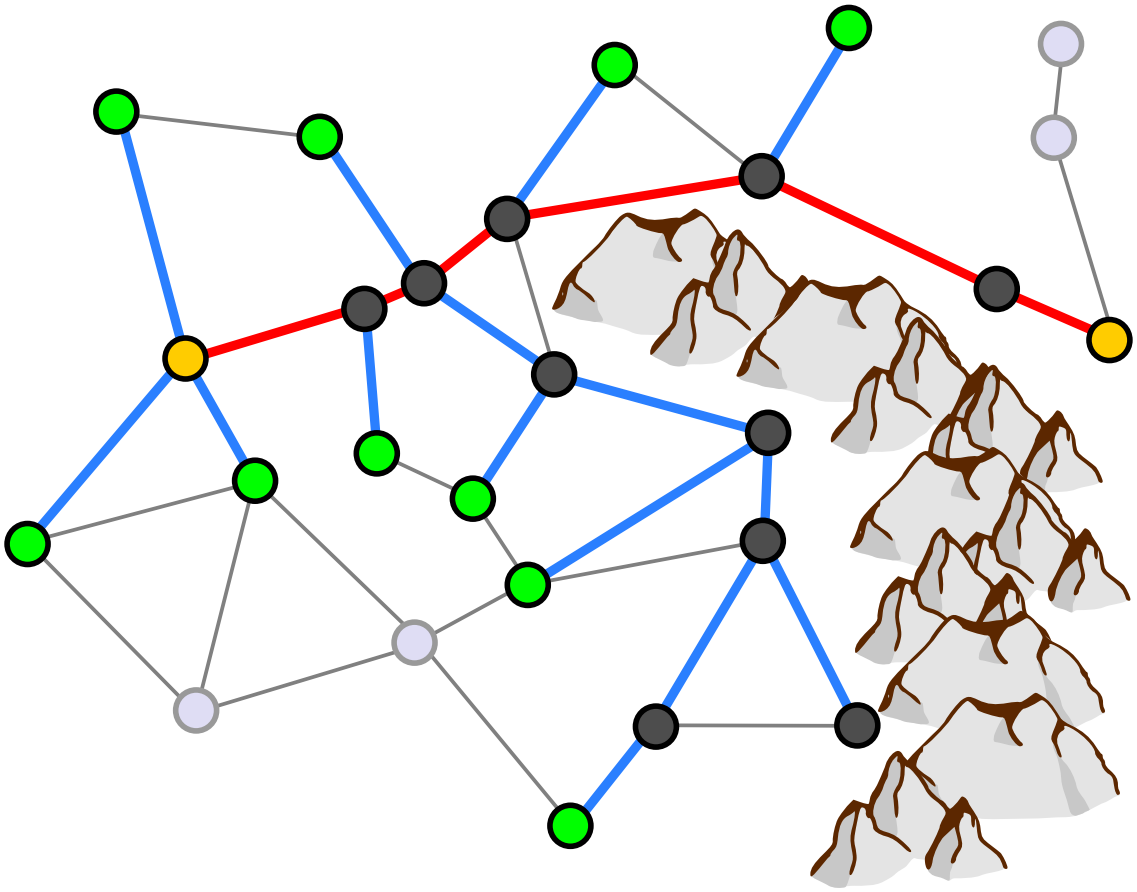
Greedy Search: Air Distance Failure



Greedy Search: Air Distance Failure

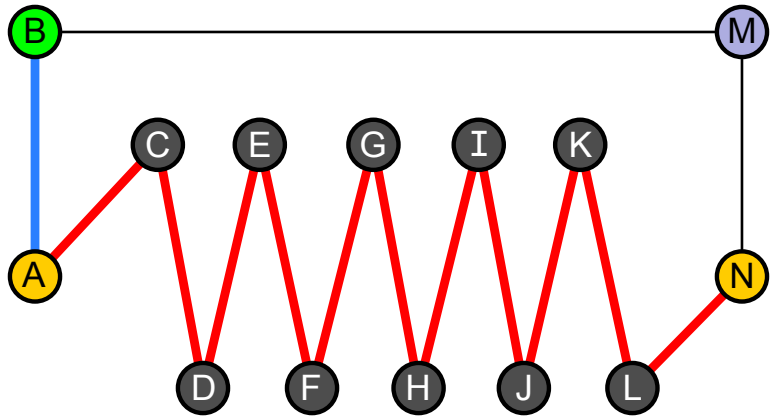


Greedy Search: Air Distance Failure



Greedy Search: Failure

The following example is showing how simply can greedy search fail in some conditions:



Greedy search will step by step expand the states C, D, E, . . . , N; state B will not be ever expanded and therefore the shortest path A, B, M, N will never be found!

A^* (A-star Search)

- Combination of greedy search and Dijkstra.
- Takes into account both the heuristic and the path cost.

Complete: Yes

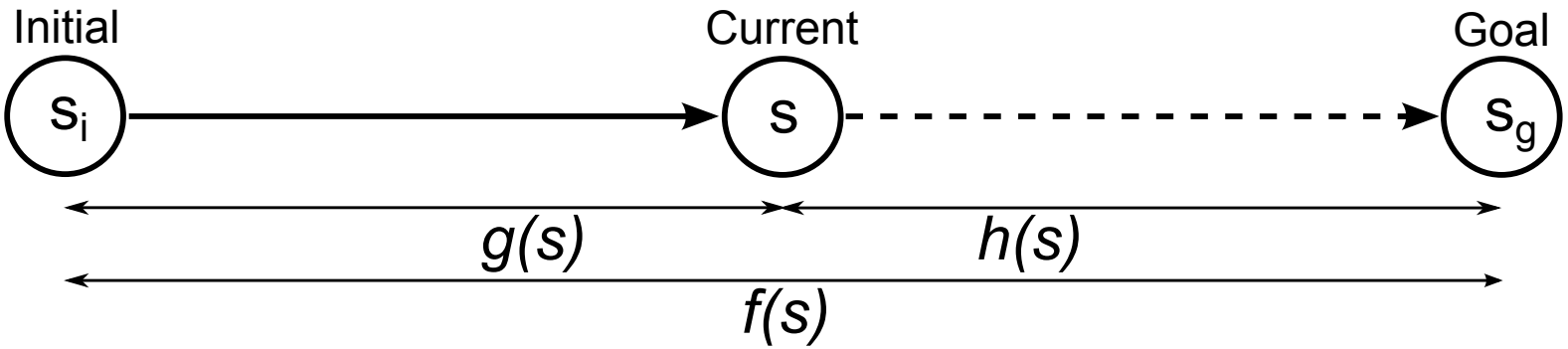
Optimal: Yes*

Time: $O(min(b^{d+1}, b|S|))$

Space: $O(min(b^{d+1}, b|S|))$

* if the heuristic h is optimistic.

Functions Notation



- $g(s)$ is a cost function that estimates least cost path from initial state to state s.
- $h(s)$ is a heuristic function that estimates the least cost path from state s to the goal state.
- $f(s)$ is an evaluation function that estimates least cost solution through state s.

A^*

Dijkstra's expands the node with the lowest cost g:

$$s^* = \arg\min_{s \in OPEN} g(s)$$

Greedy Search expands the node with lowest heuristic value:

$$s^* = \arg\min_{s \in OPEN} h(s)$$

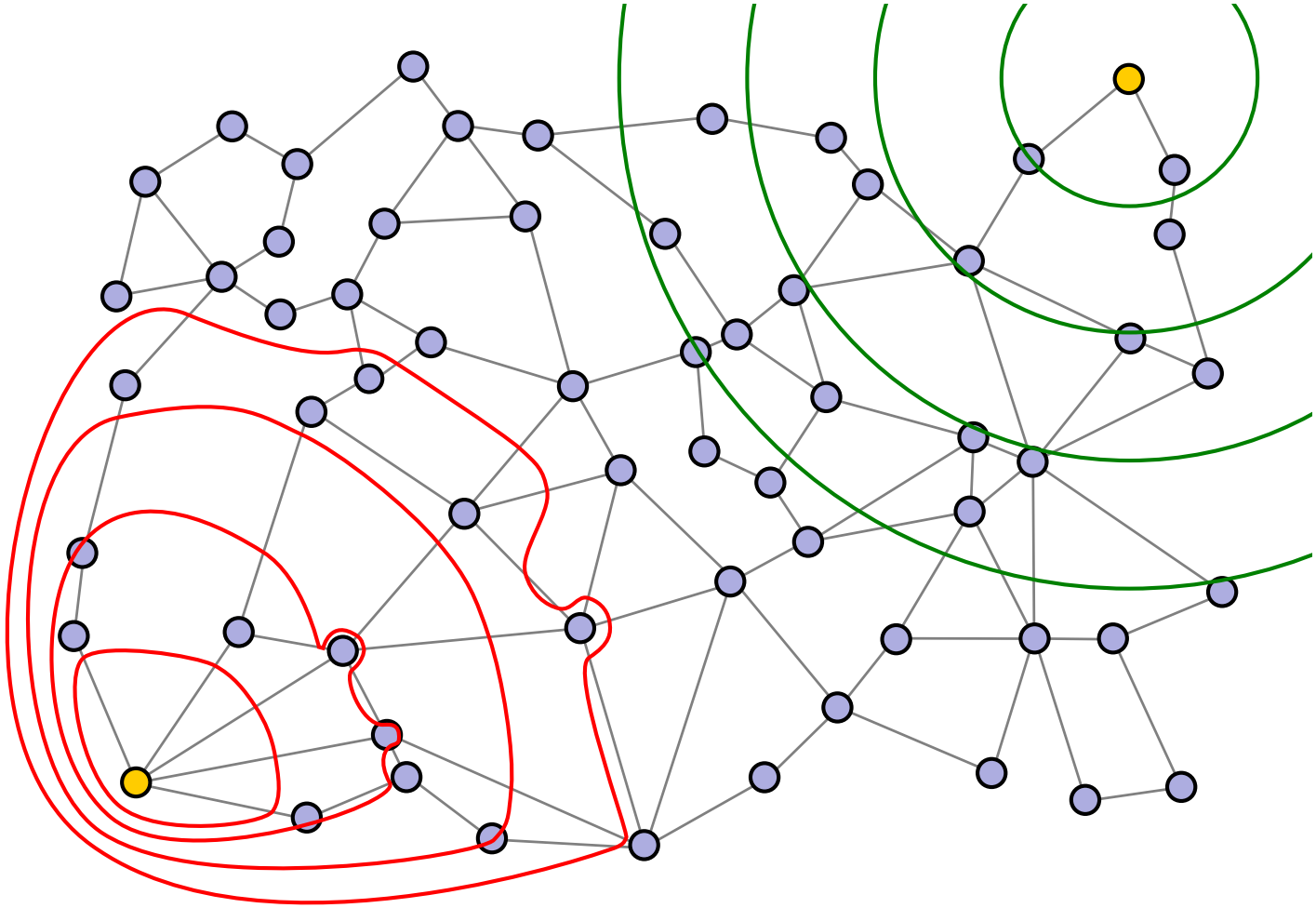
A* combines both approaches:

$$s^* = \arg\min_{s \in OPEN} (g(s) + h(s)) = \arg\min_{s \in OPEN} f(s)$$

Algorithm 3 A^*

```
 1: open ← init_priority_queue()
 2: dist ← init_table(); prev ← init_table()
 3: for all s ∈ I do
 4:     enqueue(open, s, h(s))
 5:     dist[s] ← 0; prev[s] ← null
 6: end for
 7: while ¬empty(open) do
 8:     x ← dequeue(open)
 9:     if x ∈ G then
10:         return reconstruct_path(prev, x)
11:     end if
12:     for all y ∈ neighbors(x) \ closed do
13:         d' ← dist[x] + c((x, y))
14:         if y ∉ open ∨ dist[y] > d' then
15:             dist[y] ← d'; prev[y] ← x
16:             if y ∉ open then
17:                 enqueue(open, y, d' + h(y))
18:             else
19:                 update_key(open, y, d' + h(y))
20:             end if
21:         end if
22:     end for
23:     closed ← closed ∪ {x}
24: end while
```

A^*: Real vs Heuristic Contours



Optimistic Heuristics

Definition (Optimistic heuristic)

Heuristic function h is optimistic (or admissible) if it never overestimates, i.e., its value is never greater than the true cost needed to reach the goal

$$\forall s \in S : h(s) \leq h^*(s),$$

where $h^*(s)$ is the true path cost to the goal state from state s.

- If the heuristics is not optimistic, the search algorithm might bypass the optimal path because it seems more expensive than it really is.

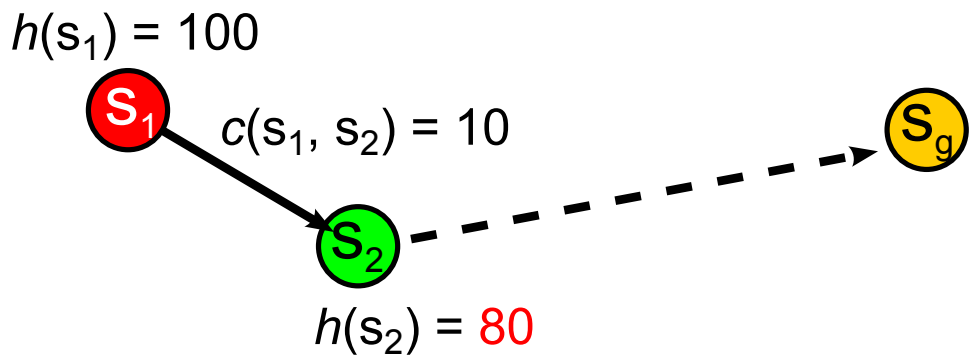
Consistent Heuristics

Definition (Consistent heuristic)

Heuristic h is consistent (or monotone) if for every state s_1 and every successor s_2 of s_1 generated by any action $a = (s_1, s_2)$, the estimated cost of reaching the goal state from s_1 is less or equal than the estimated cost of reaching the goal state from the state s_2 plus cost of the transition action a.

$$\forall (s_1, s_2) \in A : h(s_1) \leq h(s_2) + c(a)$$

Figure: Example of not consistent heuristic function (heuristic estimate decreases by 20 with move from s_1 to s_2 of cost 10)



Dominant Heuristics

Definition (Dominant heuristic)

Let A_1^* and A_2^* be optimal algorithms with optimistic heuristic functions h_1 and h_2. Algorithm A_1^* dominates algorithm A_2^* if

$$\forall s \in S : h_1(s) \geq h_2(s)$$

. We also say that A_1^* is more informed algorithm than A_2^*

- A more informed algorithm will generally search through a smaller state space than a less informed algorithm.

A^* Heuristics

Let h b a heuristic function

- if h is monotone
  - then h is also optimistic,
  - then A^* finds an optimal solution.
- if h is optimistic (but not monotone)
  - then A^* finds an optimal solution if it checks the path cost for already closed nodes.
- if h is not optimistic nor monotone
  - then A^* does not guarantee to find an optimal solution.

How To Build Heuristics

- Problem relaxation
  - the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- Combining several optimistic heuristics
  - If $h_1, h_2, \ldots, h_n$ are optimistic, we can combine them into a single dominant heuristic that will also be optimistic:

$$h(s) = \max(h_1(s), h_2(s), \ldots, h_n(s))$$

- Using sub-problem costs
  - the cost of the optimal solution of a sub-problem is a lower bound on the cost of the complete problem,
  - database of sub-problems with known cost.
- Learning heuristics
  - using machine learning techniques to derive $h(s)$ from state features, e.g. linear combination of state features $x_1, \ldots, x_n$

$$h(s) = \omega_1 x_1(s) + \omega_2 x_2(s) + \ldots \omega_n x_n(s)$$

Good Heuristic

A good heuristic is:

- **optimistic**,
- **monotone**,
- **well informed**,
- **simple to compute**.