- greedy [[Shortest Path Algorithms]] to find all destinations from one source
- For a start vertex s, compute shortest paths from s to all $v \in V$ (tree structure + length).

  Input: A connected graph $G = (V, E, w)$ with non-negative edge weights $w(u, v)$ and a vertex $s \in V$.

  Output: The distances $d(s, v)$ in G from s to all vertices $v \in V$ and the tree with the according shortest paths.
-
  - Bellman Ford is an alternative for negative weights

  Generic step: Given a set T of vertices where for all $v \in T$, $d(s, v)$ is already computed. Choose a vertex $u \in V \setminus T$ whose shortest path from s "found so far" is minimal.

  Paths "found so far": paths that only go via vertices in T.

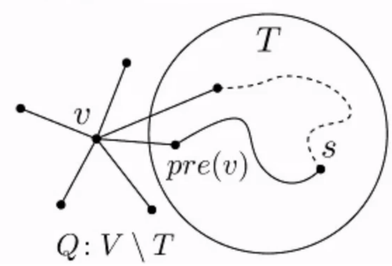
  For each vertex v, we maintain:

  $L(v)$: length of the shortest path from s to v "found so far".

  $pre(v)$: neighbor of v in T via which this shortest path goes.
-
  - similar to Prim's algorithm [[Minimum Spanning Tree]]
    - priority computation is different

  $$L(v) = \begin{cases} d(s,v) & \text{if } v \in T \\ \infty & \text{if } v \text{ is not adjacent to } T \\ \text{shortest path from } s \text{ to } v \text{ via } T & \text{if } v \notin T,\ v \text{ adjacent to } T \end{cases}$$

  A **priority queue** Q contains *all vertices that are not yet in* T, organized by their L-values (for example a min-heap; initially contains all vertices).

  -
- pseudo code

```
for all v ∈ V do L(v) = ∞ od
L(s) = 0; pre(s) = nil
Q = V                              // build up Q
while Q ≠ 0 do
   u = MIN(Q)
   remove u from Q                 // reorganize Q
   for all v ∈ A(u) do             // A: adjacency list of G
      if L(v) > L(u) + w(u,v) then
         L(v) = L(u) + w(u,v)      // reorganize Q
         pre(v) = u
```
  -

- **Runtime analysis** for graph with n vertices and m edges:
  - Min-heap with n elements:
    - $\Theta(n)$ time for initialization $Q = V$.
    - $O(\log n)$ time for removal of the minimum.
    - $O(\log n)$ time per update of an L-value.
  - Processing vertex u with $\deg(u)$ neighbors:
    removal of u from Q plus $O(\deg(u))$ updated L-values.

  $\Rightarrow$ Runtime in total for start vertex s:
  $\Theta(n) + \sum_{u \in V}(1 + \deg(u)) \cdot O(\log n)$
  $= \Theta(n) + \Theta(n + m) \cdot O(\log n) = O(m \log n)$,
  since the graph is connected.
- runtime may improve if Q is sorted
  - For dense graphs ($m = \Theta(n^2)$) the algorithm needs $\Theta(n^3 \log n)$ time to compute the distance matrix.
  - If an unsorted list is used for the queue Q, a runtime of $O(\sum_{v \in V} v \in V(n + \deg(v) \cdot 1)) = O(n^2 + m) = O(n^2)$ for start vertex s and $O(n^3)$ for the distance matrix is obtained (independent of m) $\Rightarrow$ good for dense graphs, bad for sparse graphs ($m = \Theta(n)$), works also for Prim.
- bad heuristics
  - only considers distance from start to current vertex
  - completely ignores distance from current vertex to goal
  - therefore slow
  - unlike [[A-Star Algorithm]]