1. prelimiaries

- we want to enable parallel processing by disjoint but contiguous chunks at all stages
- we start from the perspective of the points / terminals because this we can easily parallelize
- we first want to build a single-edged network graph
- we first create the joints where the perpendicular intersects
- we collect the intersection points as follows:
  - **id** = serial id to enable parallel processing in contiguous chunks downstream
    - **vertex_id** = the hilbert hash id of the POI
  - **closest_point_id** = the hilbert hash id of the perpendicular touchpoint on the closest edge
  - **closest_point_geom** = the closest point geometry. Needed, because calculating the geometry from the hilbert hash might lead to association errors later on
  - **edge_id** = the id of the closest edge

2. the process

2.1 the data procurement

2.2 the data loading

2.3 the data processing

- DONE joining the vertices to the edges
  - This can be parallelized
- OPEN creating necessary indexes on the vertex_2_edge table
  - This can not be parallelized
- DONE merging the joint_vertices into the edge geometry
  - This can be parallelized
- OPEN splitting the edge linestrings into true edges
  - This can be parallelized

2.4 the graph creation

3. progress report

2025-01-15

- developed procedure to join the vertex to the nearest edge
- challenges:
  - find the closest edge from a subset of all edges
  - solution: use a buffer to subset the set of all edges
  - challenge: how to define the buffer distance so that a closest edge candidate is found
  - solution: dynamically loop over an increasing buffer distance
  - result: now a closest edge for all vertices is found

2024-01-16

- developed procedure to merge the junctions into the edge linestring prior to chopping up into segments
- challenges:
  - manage the dependency of result of the function "st_snap()" from the tolerance value given in the function call
  - solution: loop through varying tolerance values until all junctions are preserved and merged to the linestring
  - validation: compare the desired count (junctions + points in linestring) to the result count of the "junctioned" edge
  - challenge: some junctions are points already present in the linestring (closest point = end of linestring, closest point = kink or bend in linestring)
  - solution: only count "new points" by adapting the procedure from yesterday

2025-01-17

- problem found: Assigning a serial ID to the vertex_2_edge table and chunking over the serial id can not guarantee that all occurences of an edge_id are contained in one chunk.
- solution: adapt the query for workstep 2 by adding a CTE selecting all distinct edge_ids in the given chunk and then selecting over the list of distinct edge_ids. This will require an indexing of the vertex_2_edge table on the edge_id column after workstep 1 is finished. *Create the array of distinct edge_ids with an order by statement to speed up index lookup against a btree index on edge_id. For 100.000 rows in the vertex_2_edge table the difference between no index and index is factor 7. The speedup increases with table length of course.*
-