C++ code 13.1.4: Computation of node-edge incidence matrix $\mathbf{D}_0$ ⇝ GitLab

```
Eigen::SparseMatrix<int> computeEdgeVertexIncidenceMatrix(
    const lf::mesh::Mesh &mesh) {
  // Store edge-vertex incidence matrix here
  Eigen::SparseMatrix<int, Eigen::RowMajor> G;

  // Mesh::NumEntities(unsigned codim) returns the number of elements
  // with given codimension.  Codim(Edge) = 1, Codim(Node) = 2.
  const lf::mesh::Mesh::size_type numEdges = mesh.NumEntities(1),
                                  numNodes = mesh.NumEntities(2);
  // Following the demo for the reserve()-initialising the sparse matrix
      given
  // in the exercise sheet.  From (2.1a) we know that G has exactly 2
      entries
  // per row.
  G = Eigen::SparseMatrix<int, Eigen::RowMajor>(numEdges, numNodes);
  G.reserve(Eigen::VectorXi::Constant(numEdges, 2));

  // To compute G efficiently we iterate over all edges and check the
      index
  // of the nodes at its end.  This is the efficient way to do the
      assembly,
  // introduced as "distribute scheme" in class.  We cannot iterative
      over
  // vertices, because LehrFEM++ does not allow to visit the edges
  // adjacent to a vertex
  for (const lf::mesh::Entity *edge : mesh.Entities(1)) {
    // Get index of this edge
    lf::mesh::Mesh::size_type edgeIdx = mesh.Index(*edge);
    // Get the nodes and their indices.
    // Note, that seen from the edges the nodes have codim 1, not 2,
    // hence we call SubEntities(1).  This is a relative codimension!
    auto nodes = edge->SubEntities(1);
    lf::mesh::Mesh::size_type firstNodeIdx = mesh.Index(*nodes[0]);
    lf::mesh::Mesh::size_type lastNodeIdx = mesh.Index(*nodes[1]);
    // Add the matrix entries according to the definition
    G.coeffRef(edgeIdx, firstNodeIdx) += 1;
    G.coeffRef(edgeIdx, lastNodeIdx) -= 1;
  }

  return G;
}
```