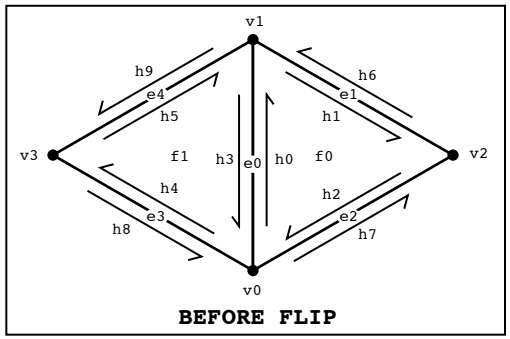
GUIDE TO IMPLEMENTING EDGE OPERATIONS ON A HALFEDGE DATA STRUCTURE

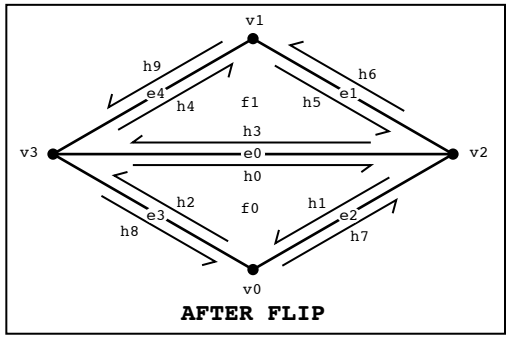
CMU CS 15-462 (FALL 2015)

PHASE 0: DRAW A DIAGRAM

First, draw a diagram of the local neighborhood both before and after the edge operation (in this case, "flip"). Label each type of element (halfedge, vertex, edge, and face) from zero to the number of elements. Make sure to include every element affected by the operation. Think very carefully about which elements will be affected - if you get things wrong in this phase, it won't matter whether the code you write in the next two phases is correct! In this example, for instance, we need to remember to include the halfedges "outside" the neighborhood, since their "twin" pointers will be affected.



BEFORE FLIP



AFTER FLIP

PHASE I: COLLECT ELEMENTS

Once you've drawn your diagram, simply collect all the elements from the "before" picture. Give them the same names as in your diagram, so that you can debug your code by comparing with the picture.

```
HALFEDGES
   HalfedgeIter h0 = e0->halfedge()
   HalfedgeIter h1 = h0->next();
   HalfedgeIter h2 = h1->next();
   HalfedgeIter h3 = h0->twin();
   HalfedgeIter h4 = h3->next();
   HalfedgeIter h5 = h4->next();
   HalfedgeIter h6 = h1->twin();
   HalfedgeIter h7 = h2->twin();
   HalfedgeIter h8 = h4->twin();
   HalfedgeIter h9 = h5->twin();

VERTICES
   VertexIter v0 = h0->vertex();
   VertexIter v1 = h3->vertex();
   ...you fill in the rest!...

EDGES
   e1 = h1->edge();
   e2 = h2->edge();
   ...you fill in the rest!...

FACES
   f0 = h0->face();
   ...you fill in the rest!...
```

PHASE Ib: ALLOCATE NEW ELEMENTS

If your edge operation requires new elements, now is the time to allocate them. For the edge flip, we don't need any new elements; but suppose for the sake of argument that we needed a new vertex v4. At this point we would allocate the new vertex via

```
   VertexIter v4 = mesh.newVertex();
```

(The name used for this new vertex should correspond to the label you give it in your "after" picture.)

PHASE II: REASSIGN ELEMENTS

Next, update the pointers for all the mesh elements that are affected by the edge operation. *Be exhaustive!* In other words, go ahead and specify *every* pointer for *every* element, even if it didn't change. Once things are working correctly, you can always optimize by removing unnecessary assignments. But get it working correctly *first!* Correctness is more important than efficiency.

```
HALFEDGES
   h0->next() = h1;
   h0->twin() = h3;
   h0->vertex() = v3;
   h0->edge() = e0;
   h0->face() = f0;

   h1->next() = h2;
   h1->twin() = h7;
   h1->vertex() = v2;
   h1->edge() = e2;
   h1->face() = f0;

   ...you fill in the rest!...
   ...and don't forget about the "outside" elements!...

   h9->next() = h9->next(); // didn't change, but set it anyway!
   h9->twin() = h4;
   h9->vertex() = v1;
   h9->edge() = e4;
   h9->face() = h9->face(); // didn't change, but set it anyway!

VERTICES
   v0->halfedge() = h2;
   v1->halfedge() = h5;
   v2->halfedge() = h3;
   v3->halfedge() = h0;

EDGES
   e0->halfedge() = h0;
   //...you fill in the rest!...

FACES
   f0->halfedge() = h0;
   //...you fill in the rest!...
```

PHASE IIb: DELETE UNUSED ELEMENTS

If your edge operation eliminates elements, now is the best time to deallocate them: at this point, you can be sure that they are no longer needed. For instance, since we don't need the vertex allocated in PHASE Ib, we could write

```
   mesh.deleteVertex( v4 );
```

You should be careful that this mesh element is not referenced by any other element in the mesh. But if your "before" and "after" diagrams are correct, this shouldn't be an issue!