**Data Structures**

- **first**: initial edge out of each node \(O(n)\)
- **to_node, next_edge**: edge information \(O(2*2m)\)
- **seen**: (Boolean array) gives nodes already seen \(O(n)\)
- **prev_edge**: implements doubly-linked list \(O(2m)\)
- **mirror**: moves between edge \((i, j)\) and \((j, i)\) \(O(2m)\)

Space complexity is \(O(2n + 8m)\), if \(G\) has \(n\) nodes and \(m\) edges.

**Overall Design** (main routine can be *self-documenting*)

1. **Initialize G** \(O(n)\)
2. **ReadGraph** \(O(m)\)
3. for \((x, y)\) in mergenodes
   - \(a = \min (x, y), \ b = \max (x, y)\) \(O(1)\)
   - initialize **seen** to indicate nodes adjacent to \(a\) \(O(n)\)
   - for all \(w \in AdjList(b)\) \(O(n)\)
     - if **seen**\((w) = \text{false}\)
       - **AddEdge** \((a, w)\), **AddEdge** \((w, a)\)
     endif
   endfor
4. **RemoveEdge** \((w, b)\)
5. endfor
6. **AdjList(b) = \[]** \(O(1)\)
7. **PrintGraph** \(O(n + m)\)

Time complexity is \(O(n + m)\) for SETUP and \(O(n)\) per COALESCE if we can carry out the commands within the inner for loop in constant time. For a sequence of \(O(n)\) COALESCE operations, runs in \(O(n + m + n^2) = O(n^2)\) time.
Code Issues

**Initialize G:** \( \text{first}[i] = 0 \); this avoids special cases later.

**ReadGraph:** one by one, read edge data; insert new edges at front of linked adjacency list. No need for a last pointer.

Develop a module **AddEdge** \((u, v)\) to add this directed edge to the data structure; call twice for adding undirected edges in **ReadGraph**. This can be used in the COALESCE routine (constant time).

How to carry out **RemoveEdge** \((w, b)\) in constant time?

Since we have edge \((b, w)\) already, use **mirror** to take us to \((w, b)\).
Then use doubly-linked list to delete \((w, b)\) in constant time.

**mirror** can also be implemented using arithmetic for a static graph.

To avoid the loop \((a, a)\) when \(b\) is adjacent to \(a\), we can set **seen** \((a) = \text{true}\); this will avoid special cases.

Can reuse space for edges \((b, w), (w, b)\) when adding edges \((a, w), (w, a)\).

**PrintGraph:** run through adjacency lists of all remaining nodes.

```c
AddEdge(edgeloc, head, tail)
    to_node[edgeloc] = tail;
    next_edge[edgeloc] = first[head];
    first[head] = edgeloc;
```

**General Comments**

Proper initialization can avoid checking special cases each time.

Modularize your code.

Give variables, arrays informative/descriptive names.