In-Class Exercise

- Input: social network graph stored as two files
  - Nodes: (userID, name, hobbies)
  - Edges: (userID, friendID)
- Find all cricket players in “my” 3-hop neighborhood

- Pre-process: create adjacency list for each person, add this to node object

```
// Initially only start node “me” is active. It is not reached and will remain so.
map(nid n, nodeObject N) {
 if (n <> “me”) emit(n, N)  // Do not process start node any more

 if (N.isActive()) {
  N.unsetActive() // De-activate node to avoid repeat processing
  for all nid m in N.adjacencyList do
   if (m <> “me”) emit(m, NULL)
 }
}
```

```
// Identify newly reached nodes and set them to reached and active.
reduce(nid m, [n1, n2,…]) {
 isReached = false; M = NULL

 for all n in [n1,n2,…] do
  if isNode(n) then // The node object was found: recover graph structure
   M = n
  else // This node was reached from an active node. Set as reached.
   isReached = true

 // Set this node as active if it is reached for the first time
 if (not M.wasReached()) {
  M.setReached(); M.setActive()
 }

  emit(nid m, node M)
}
```

Filter Condition

- The program above is called 3 times.
  - Final result: 3-hop friends = nodes with wasReached() = true
- Map-only parallel scan of this result to find the cricket players
- Avoid forwarding reached nodes in Mapper?
  - Ok: handle NULL node object in reduce call, do not emit it
  - But: reached nodes are scattered over output of different iterations

In-Class Exercise

- Input: same social network graph
  - Nodes annotated with userID, name, hobbies
  - Edges = friendship links
- Find all cricket players in “my” 3-hop neighborhood and
- *Return them sorted by friendship distance*

Solution 1

- Pre-process: create adjacency list for each person, add this to node object
- Remember in which iteration the node was *first* reached
  - Pass iteration counter into context (global constant)
  - In Reduce, set a new distance variable to the iteration counter in the "if not M.wasReached()" block
- Run a sort post-processing step with distance variable as key, where map eliminates nodes that were not reached

Solution 2

- Pre-process: create adjacency list for each person, add this to node object
- Do not emit reached nodes in Mapper
- Friends discovered in i-th iteration are in output of i-th job
- Just pick them up in order from the corresponding output directories

```
// Initially only start node “me” is active. It is not reached and will remain so.
map(nid n, nodeObject N) {
 if (n <> “me” AND !N.wasReached()) emit(n, N)  // Do not process start node and reached nodes again

 if (N.isActive()) {
  N.unsetActive() // De-activate node to avoid repeat processing
  for all nid m in N.adjacencyList do
   if (m <> “me”) emit(m, NULL)
 }
}
```

```
// Identify newly reached nodes and set them to reached and active.
reduce(nid m, [n1, n2,…]) {
 isReached = false; M = NULL

 for all n in [n1,n2,…] do
  if isNode(n) then // The node object was found: recover graph structure
   M = n
  else // This node was reached from an active node
   isReached = true

 // If the node object is NULL, the node was reached before: ignore it
 if (M <> NULL) {
  if (isReached) { M.setReached(); M.setActive()}

  emit(nid m, node M)
 }
}
```