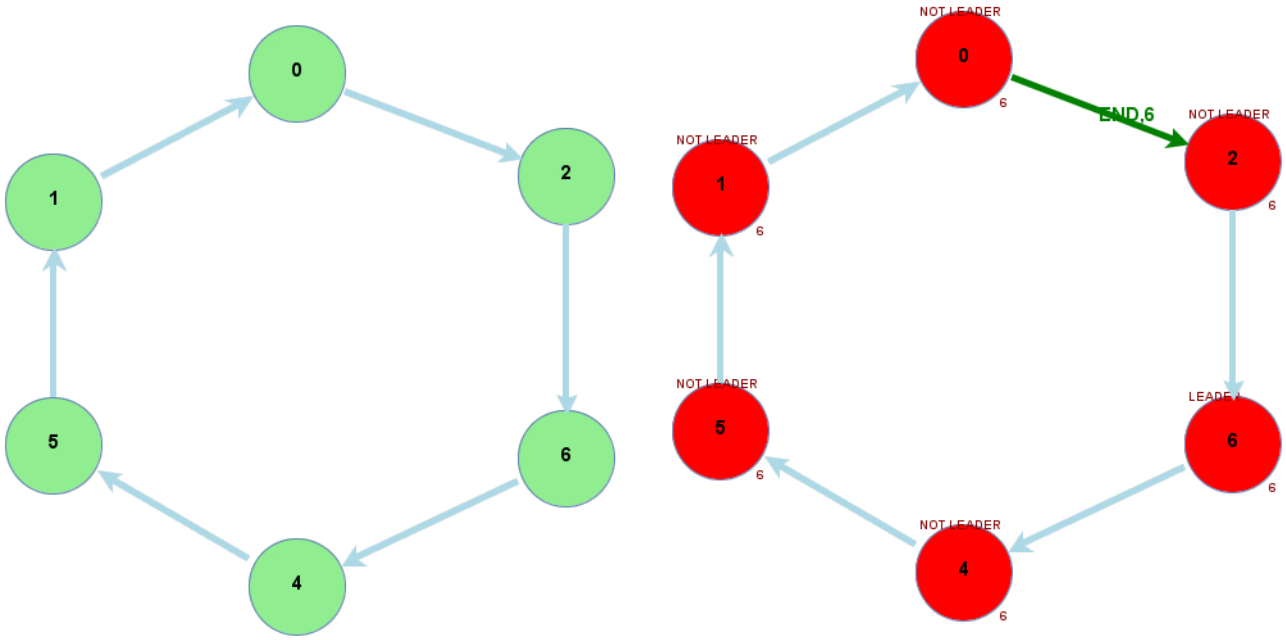
Appendix C: Example Algorithms

Leader Election

The algorithm in Listing 2 elects a leader in a directional ring network as shown in Figure 12a. It is assumed that a directional ring network of at least two processors exists and that there is only one link from each processor. It is required that all processors in the network have numerical IDs.



(a) Example directional ring network.

(b) Example output.

Figure 12: Example input and output for leader election algorthim.

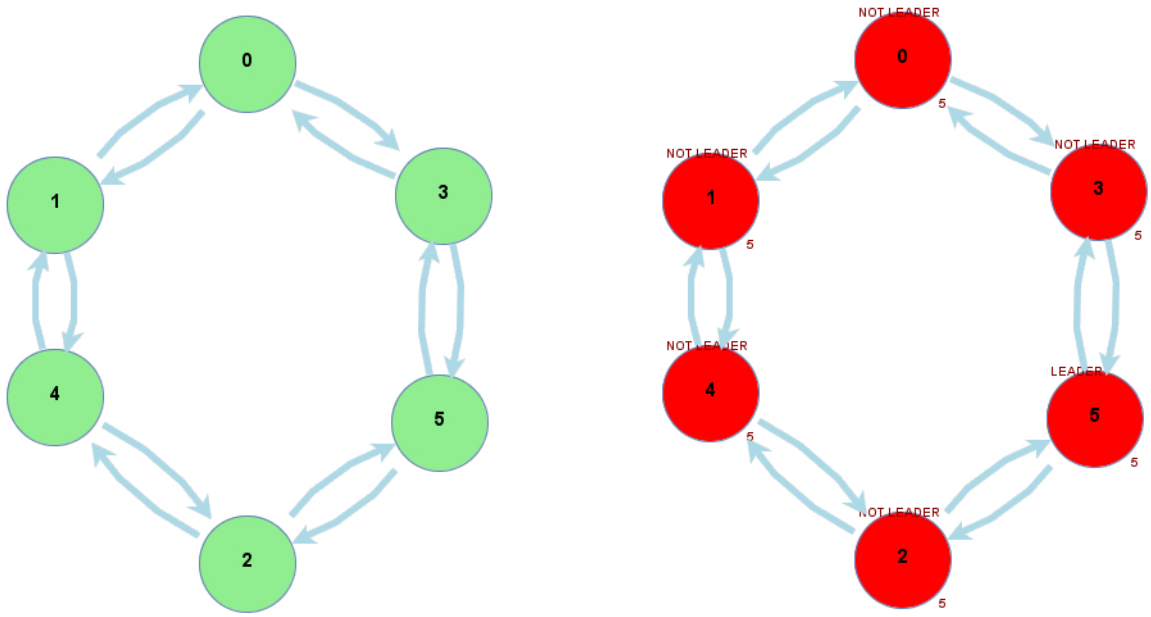
Listing 2: Directional Ring Leader Election

```
1  package dans.algorithm.examples;
2
3  import dans.algorithm.Algorithm;
4  import dans.algorithm.Message;
5
6
7  public class LeaderElection extends Algorithm {
8
9      @Override
10     public Object algorithm() {
11         int id = Integer.parseInt(getID());
12         String msg = getID();
13         String status = "UNKOWN";
14         String neighbors[] = getNeighbors();
15         int leader = -1;
16
17         while(doMainLoop()) {
18             if (msg ≠ null) {
19                 send(neighbors[0], msg);
20
21                 if(msg.startsWith("END")) {
```

```
22                      return leader;
23                  }
24              }
25
26              msg = null;
27
28              Message m = receive(false);
29              if(m ≠ null) {
30                  if(m.message().startsWith("END")) {
31                      leader = Integer.parseInt(m.message().split(",")[1]);
32
33                      if(neighbors[0].equals(leader+"")) {
34                          return leader;
35                      } else {
36                          msg = m.message();
37                      }
38                  } else {
39                      int i = m.toInt();
40                      if(i > id) {
41                          status = "NOT LEADER";
42                          msg = m.message();
43                      } else if(id > i) {
44                          msg = null;
45                      } else {
46                          leader = Integer.parseInt(getID());
47                          status = "LEADER";
48                          msg = "END," + leader;
49                      }
50                  }
51              }
52
53              display(status);
54          }
55
56          return leader;
57      }
58
59  }
```

Bidirectional Leader Election

The algorithm in Listing 3 elects a leader in a bidirectional ring network as shown in Figure 13a. It is assumed that a bidirectional ring network of at least two processors exists and that each processor has a link to the neighbour on its left and right in the ring. It is further assumed that the processors do not know which neighbour is their left neighbour and which is their right neighbour just that they know the IDs of two neighbours. It is required that all processors in the network have numerical IDs.



(a) Example bidirectional ring network.

(b) Example output.

Figure 13: Example input and output for leader bidirectional election algorthim.

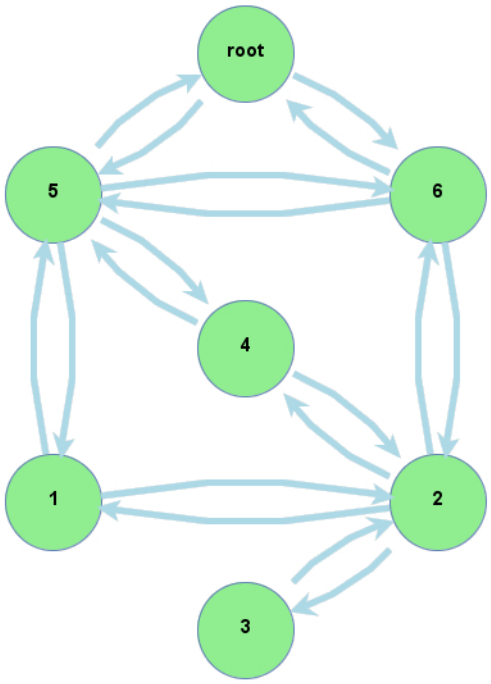
Listing 3: Bidirectional Ring Leader Election

```
1  package dans.algorithm.examples;
2
3  import dans.algorithm.Algorithm;
4  import dans.algorithm.Message;
5
6
7  public class BidirectionalLeaderElection extends Algorithm {
8
9      @Override
10     public Object algorithm() {
11         int id = Integer.parseInt(getID());
12         String msgA = getID();
13         String msgB = getID();
14         String status = "UNKOWN";
15         String neighbors[] = getNeighbors();
16         int leader = -1;
17
18         while(doMainLoop()) {
19             if(msgA ≠ null) {
20                 send(neighbors[0], msgA);
21             }
22
23             if(msgB ≠ null) {
```

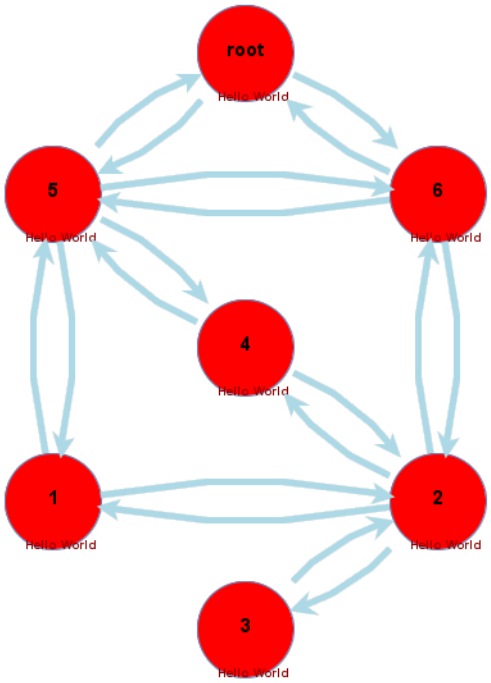
```
                send(neighbors[1], msgB);
            }

            if(leader >= 0) {
                return leader;
            }

            msgA = null;
            msgB = null;

            Message in;
            while((in = receive()) ≠ null) {
                String m = in.message();
                String from = in.from();
                String splited[] = m.split(",");

                if(m.startsWith("END")) {
                    leader = Integer.parseInt(splited[1]);

                    if(from.equals(neighbors[0])) {
                        if(leader == Integer.parseInt(neighbors[1])) return leader;
                        msgB = m;
                    } else {
                        if(leader == Integer.parseInt(neighbors[0])) return leader;
                        msgA = m;
                    }
                } else {
                    int i = in.toInt();

                    if(i > id) {
                        status = "NOT LEADER";

                        if(from.equals(neighbors[0])) {
                            msgB = m;
                        } else {
                            msgA = m;
                        }
                    } else if(i == id) {
                        status = "LEADER";
                        leader = id;
                        msgA = "END," + id;
                    }
                }

                display(status);
            }
        }

        return leader;
    }

}
```

Broadcast

The algorithm in Listing 4 broadcasts the message "Hello World" from the processor with the ID "root" to all other processors in the network. It is assumed that all links are bidirectional and that the network graph is strongly connected. It is required that one and only one processor be given the ID "root". An example network and corresponding output are shown in Figure 14.



(a) Example network for broadcasting.



(b) Example output.

Figure 14: Example input and output for asynchronous broadcasting algorthim.

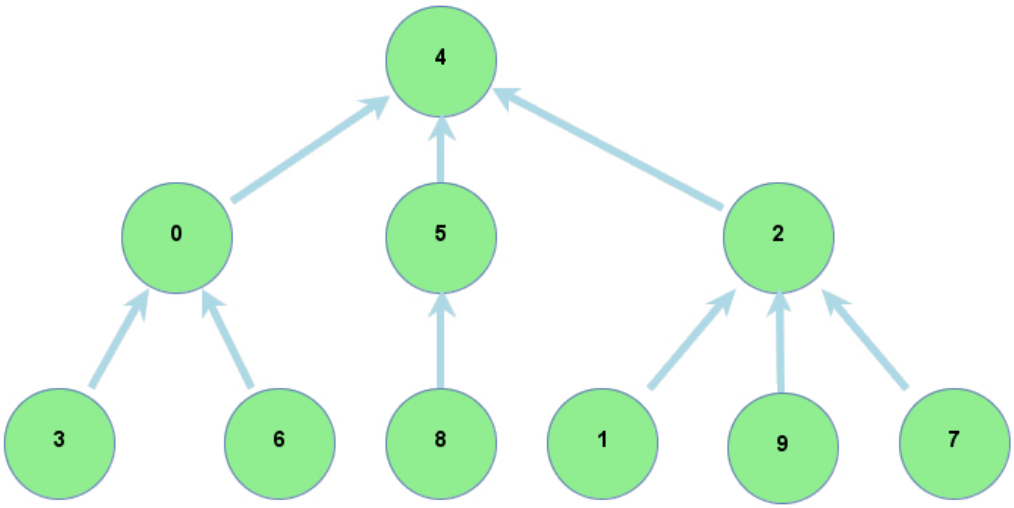
Listing 4: Asynchronous Broadcast

```
1  package dans.algorithm.examples;
2
3  import dans.algorithm.Algorithm;
4  import dans.algorithm.Message;
5  import java.util.ArrayList;
6
7  public class Broadcast extends Algorithm {
8      @Override
9      public Object algorithm() {
10         String parent = null;
11         String message = null;
12         int numNeighbors = getNeighbors().length;
13         ArrayList<String> acked = new ArrayList<>();
14
15         if(getID().equalsIgnoreCase("root")) {
16             message = "Hello World";
17             for(String nid : getNeighbors()) {
18                 send(nid, "BROADCAST," + message);
19             }
20         }
21
22         while(doMainLoop()) {
23             Message m;
24             while((m = receive()) ≠ null) {
```

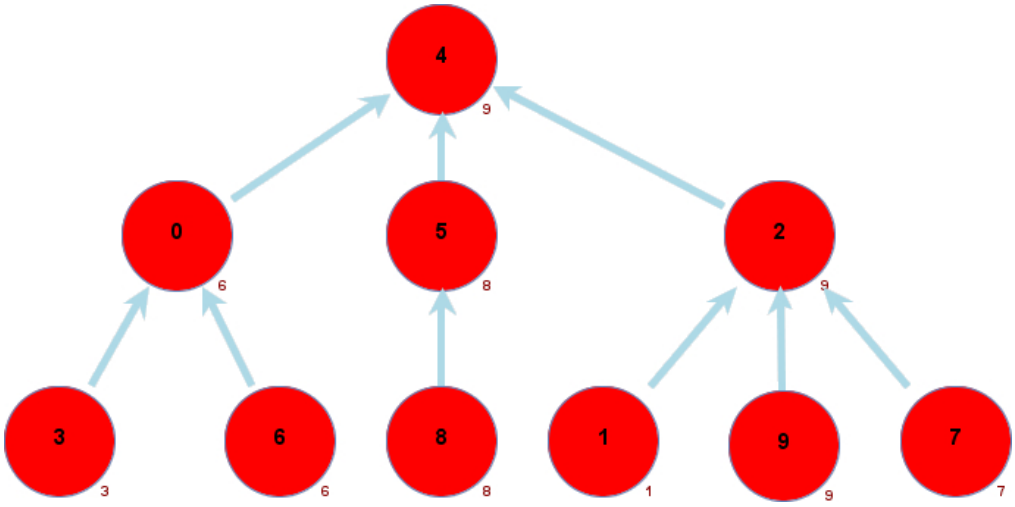
```
25                  if(m.message().startsWith("BROADCAST,")) {
26                      if(message == null) {
27                          message = m.message().split(",")[1];
28                          parent = m.from();
29                          for(String nid : getNeighbors()) {
30                              if(!nid.equals(parent)) {
31                                  send(nid,  m.message());
32                              }
33                          }
34                      } else {
35                          send(m.from(),"ACK");
36                      }
37                  } else if(m.message().equals("ACK")) {
38                      acked.add(m.from());
39                  }
40              }
41
42              if(getID().equalsIgnoreCase("root")) {
43                  if(acked.size() >= numNeighbors) {
44                      return message;
45                  }
46              } else {
47                  if(parent ≠ null && acked.size() >= numNeighbors - 1) {
48                      send(parent, "ACK");
49                      return message;
50                  }
51              }
52
53          }
54
55          return message;
56      }
57
58  }
```

Largest ID

Given a BFS tree of a network, the algorithm in Listing 5 finds the largest ID of any processor in the network. It is assumed that the BFS tree is represented by laying the network graph out in a tree with only one directional link leaving each processor (other than the root) as shown in Figure 15a. Only the root processor will find the correct largest ID, any other processor will simply have the largest ID between its self and its children. It is required that all processor IDs be numerical.



(a) Example tree network for finding largest ID.



(b) Example output.

Figure 15: Example input and output for find largest ID algorthim.

Listing 5: Find Largest ID

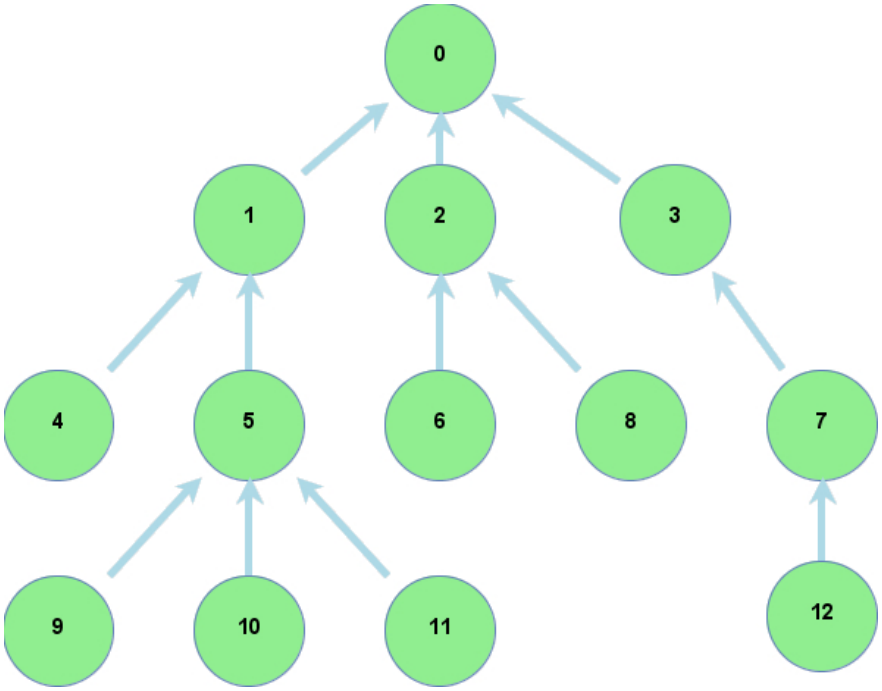
```
package dans.algorithm.examples;

import dans.algorithm.Algorithm;
import dans.algorithm.Message;


public class TreeLargestID extends Algorithm {

    @Override
    public Object algorithm() {
        String parents[] = getNeighbors();
        String childern[] = getSources();
        int id = Integer.parseInt(getID());
        String msg = null;
        int largest = id;
        int heardfrom = 0;


        if(childern.length == 0) {
            msg = getID();
        } else if(parents.length == 0 && childern.length == 0) {
            return largest;
        }

        while(doMainLoop()) {
            if(msg ≠ null) {
                for(String pid : parents) {
                    send(new Message(pid, msg));
                }

                return largest;
            }

            msg = null;

            Message in;
            while((in = receive()) ≠ null) {
                int i = in.toInt();

                if(i > largest) {
                    largest = i;
                }

                heardfrom++;
            }

            if(heardfrom  >= childern.length) {
                msg = largest + "";
            }
        }

        return largest;
    }

}
```

ID Sum

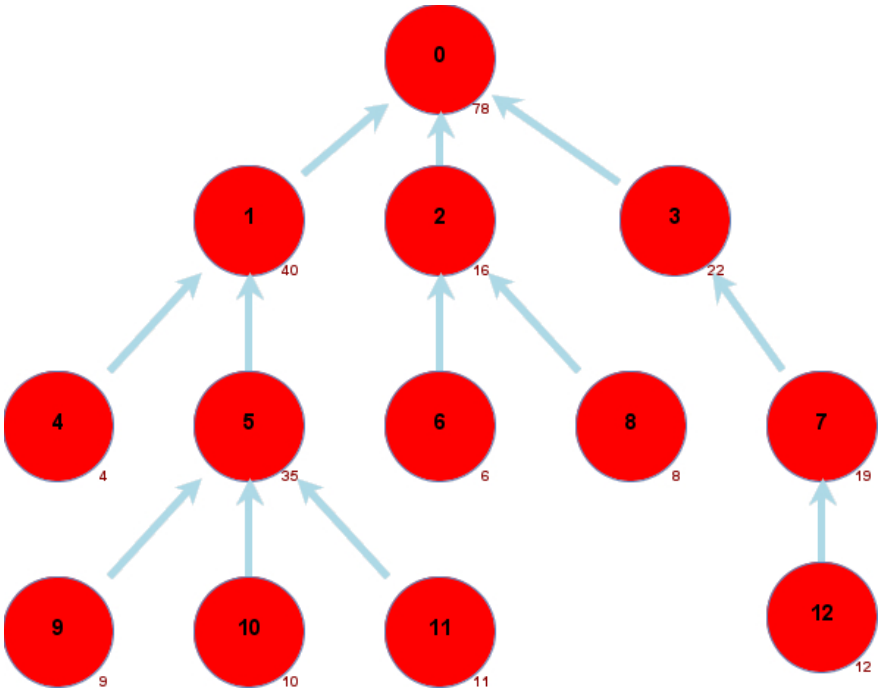
Given a BFS tree of a network, the algorithm in Listing 6 finds the sum of the processor's IDs in the network. It is assumed that the BFS tree is represented by laying the network graph out in a tree with only one directional link leaving each processor (other than the root) as shown in Figure 16a. Only the root processor will find the correct sum, any other processor will simply have the sum of its self and its children. It is required that all processor IDs be numerical.

Listing 6: Find Sum of IDs

```
1  package dans.algorithm.examples;
2
3  import dans.algorithm.Algorithm;
4  import dans.algorithm.Message;
5
6
7  public class TreeSumID extends Algorithm {
8
9      @Override
10     public Object algorithm() {
11         String parents[] = getNeighbors();
12         String childern[] = getSources();
13         int id = Integer.parseInt(getID());
14         String msg = null;
15         int sum = id;
16         int heardfrom = 0;
17
18
19         if(childern.length == 0) {
20             msg = getID();
21         } else if(parents.length == 0 && childern.length == 0) {
22             return sum;
23         }
24
25         while(doMainLoop()) {
26             if(msg ≠ null) {
27                 for(String pid : parents) {
28                     send(new Message(pid, msg));
29                 }
30
31                 return sum;
32             }
33
34             msg = null;
35
36             Message in;
37             while((in = receive()) ≠ null) {
38                 int i = in.toInt();
39                 sum += i;
40
41                 heardfrom++;
42             }
43
44             if(heardfrom  >= childern.length) {
45                 msg = sum + "";
46             }
47         }
48
49         return sum;
50     }
51
52 }
```



(a) Example tree network for finding the sum of IDs.



(b) Example output.

Figure 16: Example input and output for find sum of IDs algorthim.