Práctica 2
Algoritmos

Labrador Mata Janet
Luna Villanueva Karla Victoria
Olivos Noriega Danna Abigail

Octubre 2025

1 NodoVecinos.py

El Algoritmo 1 - Conocer vecinos de vecinos, como su nombre lo dice, nos permite que cada nodo conozca no solo a sus vecinos directos, sino también a los vecinos de sus vecinos directos.

```
1: id_i = i
2: neighbors_i = { Conjunto de vecinos de p_i }
3: identifiers_i = {∅}

4: main()
5: begin:
6:     for each j ∈ neighbors_i do
7:         send MYNAME(neighbors_i) to j
8:     end for
9: end

10: when MYNAME(identifiers_j) is received from neighbor p_j
11: begin:
12:     identifiers_i = identifiers_i ∪ identifiers_j
13: end
```

Figure 1: Algoritmo 1

- **Funcionamiento:**
  1. Cada nodo espera 1 TICK
  2. Envía su lista de vecinos a todos sus vecinos directos
  3. Cada nodo recibe mensajes de todos sus vecinos directos
  4. Actualiza su conjunto **identifiers** con los IDs recibidos
  5. Al final, cada nodo conoce todos los nodos a distancia 2

2 NodoGenerador.py

El Algoritmo 4 - Construcción del árbol generador, construye un árbol generador donde el nodo 0 es la raíz y se establecen relaciones padre-hijo con los nodos.

```
1: Initially do
2: begin:
3: if ps = pi then
4:     parent_i = i; expected_msg_i = |neighbors_i|
5:     for each j ∈ neighbors_i do send GO() to pj
6:     end for
7: else parent_i = ∅
8: end if
9: children_i = ∅
10: end

11: when GO() is received from pj do
12: begin:
13: if parent_i = ∅ then
14:     parent_i = j; expected_msg_i = |neighbors_i| - 1
15:     if expected_msg_i = 0 then send BACK(i) to pj
16:     else
17:         for each k ∈ neighbors_i \ {j} do send GO() to pk
18:         end for
19:     end if
20: else send BACK(∅) to pj
21: end if
22: end

23: when BACK(val set) is received from pj do
24: begin:
25: expected_msg_i = expected_msg_i - 1
26: if val set ≠ ∅ then children_i = children_i ∪ {j}
27: end if
28: if expected_msg_i = 0 then
29: if parent_i ≠ i then
30:     send BACK(i) to parent_i
31: end if
32: end if
33: end
```

Figure 2: Algoritmo 4

- **Estados del nodo:**
  - **padre:** referencia al nodo padre en el árbol
  - **hijos:** lista de nodos hijos en el árbol
  - **expected_msg:** número de mensajes BACK esperados
  - **es_raiz:** indica si el nodo es la raíz o no
  - **procesado:** indica si el nodo ya fue procesado
- **Funcionamiento:**
  1. Inicio con el nodo raíz:
     - Se marca como procesado
     - Se establece como su propio padre
     - Calcula mensajes BACK esperados
     - Envía mensajes GO a todos sus vecinos
  2. Recepción del GO:
     - Si el nodo no tiene padre y no está procesado:
       - Establece al emisor como padre
       - Calcula mensajes BACK esperados
       - Si tiene otros vecinos, les envía GO
       - Si no tiene otros vecinos, envía BACK al padre
     - Si ya tiene padre, envía BACK vacío al emisor
  3. Recepción de BACK:
     - Si el BACK contiene un ID válido, lo agrega a sus hijos
     - Decrementa el contador de mensajes esperados
     - Si ya recibió todos los BACK esperados y no es raíz, envía BACK a su padre

3 NodoBroadcast.py

El Algoritmo 5 - Broadcast, permite que un mensaje sea difundido desde un nodo distinguido (nodo 0) a todos los demás nodos.

```
1: Initially do
2: begin:
3: if ps = pi then
4:     data = mensaje que se quiere difundir
5:     for each j ∈ children_i do send GO(data) to pj
6:     end for
7: else data = ∅
8: end if
9: end

10: when GO(data) is received from pj do
11: begin:
12: for each k ∈ children_i do send GO(data) to pk
13: end for
14: end
```

Figure 3: Algoritmo 5

- **Funcionamiento:**
  - **Nodo distinguido(nodo 0):**
    1. Espera 1 unidad de tiempo (TICK)
    2. Envía el mensaje a todos sus vecinos inmediatos
  - **Nodos no distinguidos:**
    1. Esperan a recibir el mensaje por el canal de entrada
    2. Almacenan el mensaje recibido
    3. Esperan 1 unidad de tiempo (TICK)
    4. Reenvían el mensaje a todos sus vecinos
    5. Terminan su ejecución después de reenviar