Computación Distribuida 2024-1

Práctica 3 – BFS y DFS sin terminación

Profesor: Mauricio Riva Palacio Orozco

Ayudantes: Adrián Felipe Fernández Romero y Alan Alexis Martínez López

1. Descripción de la práctica

En esta práctica deberán implementar los algoritmos DFS y BFS en sus versiones que no detectan terminación. El pseudocódigo de referencia se encuentra en la carpeta 'practica 3'.

2. Desarrollo

La práctica está conformada por tres archivos principales (con sus respectivas interfaces) y un archivo de prueba:

- NodoBFS.py
- NodoDFS.py
- CanalRecorridos.py
- Test.py

En cada uno deberán implementar la semántica de los nodos de los algoritmos correspondientes, siguiendo la estructura que se les proporciona.

3. Prueba (Test.py)

Estas pruebas tienen la finalidad de validar los resultados de sus implementaciones. Ojo: que las pruebas unitarias pasen con éxito no garantiza la calificación máxima. Tómenlas únicamente como apoyo para el diseño y depuración de sus algoritmos.

3.1. Prerrequisitos

Para ejecutar las pruebas basta con correr en la terminal el siguiente comando:

```
pytest -q test.py
```

4. Observaciones

- Respeta los constructores proporcionados.
- Por convención, el nodo distinguido en todos los algoritmos será el nodo con índice 0.
- En BFS: a los atributos del nodo i se les llamará padre (id del padre) y distancia (distancia al nodo distinguido).
- En DFS: a los atributos se les llamará padre y vecinos.
- En BFS y DFS, al crear un nodo, por convención la referencia al padre será el mismo nodo.
- En DFS, debido a su naturaleza no determinista, haremos una modificación: en lugar de elegir un vecino aleatorio, el algoritmo siempre seleccionará el vecino con el menor identificador.

Ejemplo:

$$N_{nv}(v_i) = \{v_3, v_6, v_2, v_7, v_9, v_8\}$$

Entonces el vértice elegido para continuar la ejecución del algoritmo sería v2.

5. Lineamientos de entrega

Cada clase debe estar bien documentada. Se debe incluir un archivo ReadMe con el número de la práctica, los nombres y números de cuenta de los integrantes y una explicación de la implementación de cada algoritmo.

Solo un integrante debe subir la práctica; el otro debe marcarla como entregada. La práctica debe comprimirse en un archivo .zip con el nombre 'Practica3'.

En caso de dudas, comunicarse con el equipo de ayudantes.

Anexo A: Gráficas de prueba

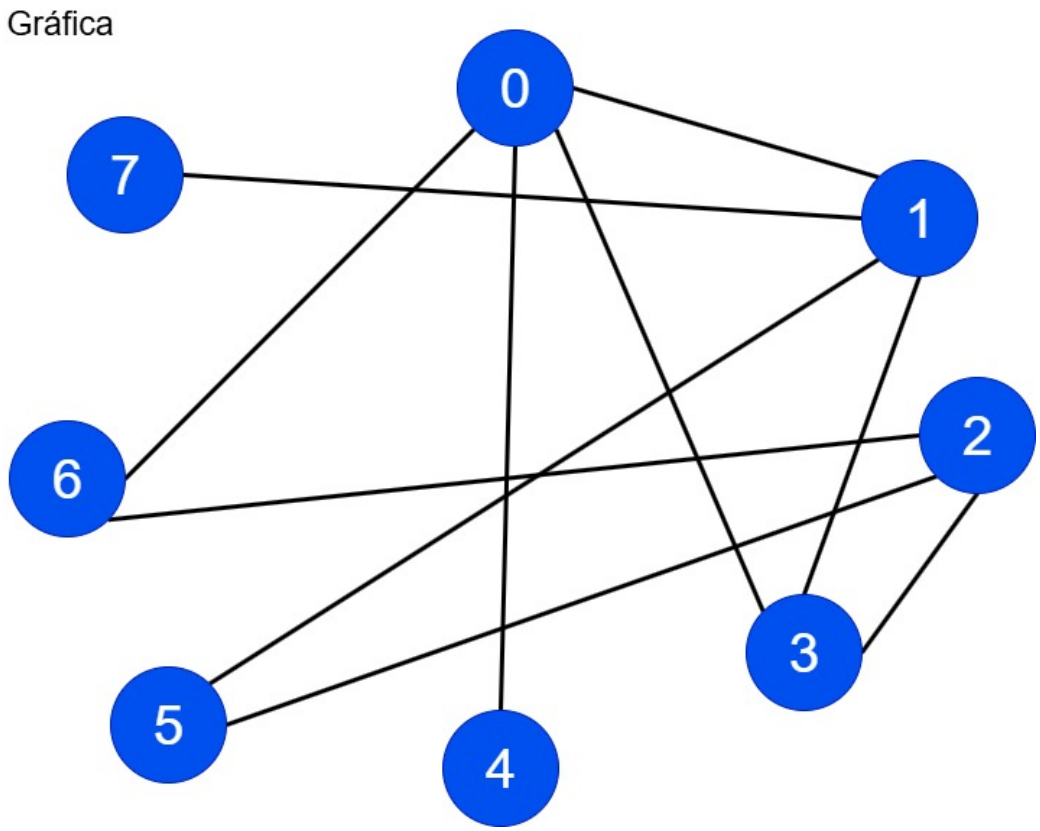

Figura 1. Gráfica general

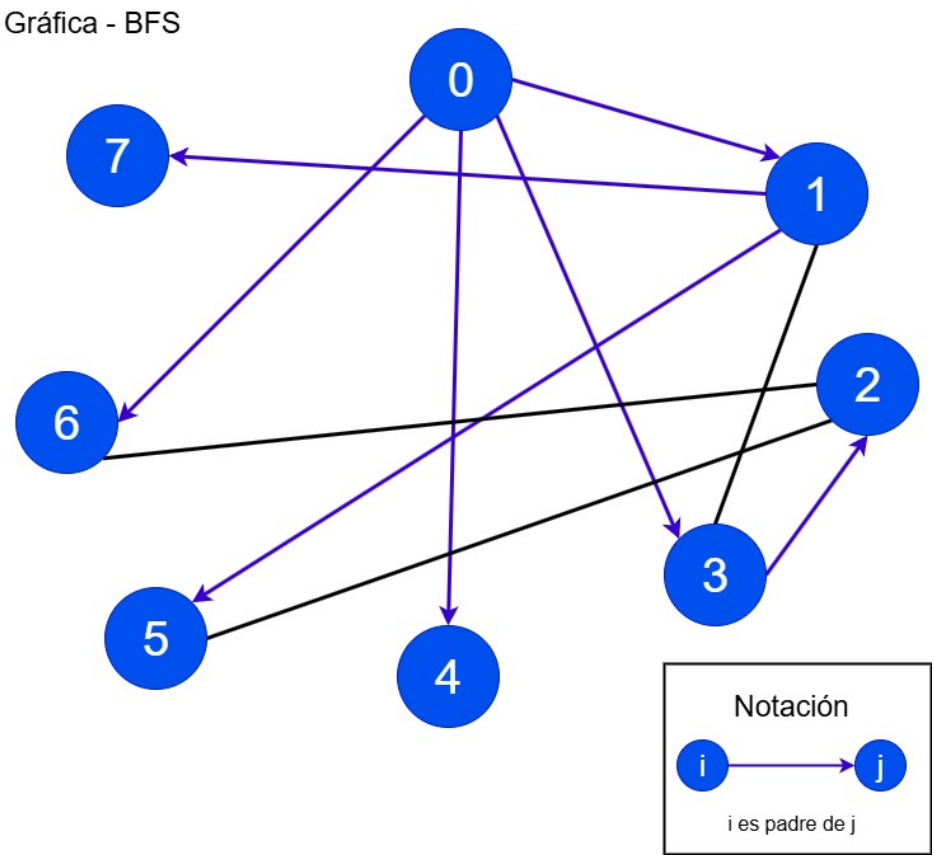

Figura 2. Árbol BFS

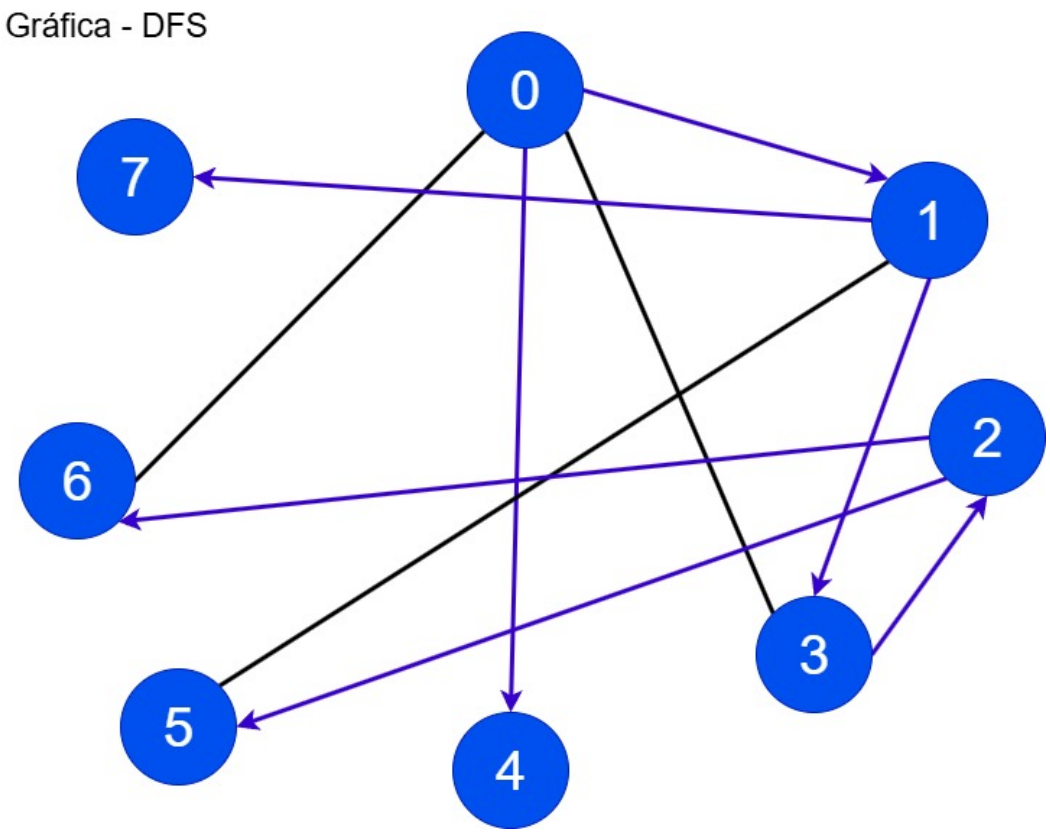

Figura 3. DFS

OBSERVACIÓN. Algoritmo BFS

```
when START() is received do    % only the distinguished process receives this message %
(1)   send GO(−1) to itself.

when GO(d) is received from p_j do
(2)   if (parent_i = ⊥)
(3)      then parent_i ← j; children_i ← ∅; level_i ← d + 1;
(4)           expected_msg_i ← |neighbors_i \ {j}|;
(5)           if (expected_msg_i = 0)
(6)              then send BACK(yes, d + 1) to p_parent_i
(7)              else for each k ∈ neighbors_i \ {j} do send GO(d + 1) to p_k end for
(8)           end if
(9)      else if (level_i > d + 1)
(10)             then parent_i ← j; children_i ← ∅; level_i ← d + 1;
(11)                  expected_msg_i ← |neighbors_i \ {j}|;
(12)                  if (expected_msg_i = 0)
(13)                     then send BACK(yes, level_i) to p_parent_i
(14)                     else for each k ∈ neighbors_i \ {j} do send GO(d + 1) to p_k end for
(15)                  end if
(16)             else send BACK(no, d + 1) to p_j
(17)          end if
(18)  end if.

when BACK(resp, d) is received from p_j do
(19)  if (d = level_i + 1)
(20)     then if (resp = yes) then children_i ← children_i ∪ {j} end if;
(21)          expected_msg_i ← expected_msg_i − 1;
(22)          if (expected_msg_i = 0)
(23)             then if (parent_i ≠ i) then send BACK(yes, level_i) to p_parent_i
(24)                                    else p_i learns that the breadth-first tree is built
(25)                  end if
(26)          end if
(27)  end if.
```

Fig. 1.11 Construction of a breadth-first spanning tree without centralized control (code for p_i)

Figura 4. Algoritmo BFS (fuente original)

```
when START() is received do    % only p_a receives this message %
(1)  parent_i ← i;
(2)  let k ∈ neighbors_i;
(3)  send GO({i}) to p_k; children_i ← {k}.

when GO(visited) is received from p_j do
(4)  parent_i ← j;
(5)  if (neighbors_i ⊆ visited)
(6)     then send BACK(visited ∪ {i}) to p_j; children_i ← ∅;
(7)     else let k ∈ neighbors_i \ visited;
(8)          send GO(visited ∪ {i}) to p_k; children_i ← {k}
(9)  end if.

when BACK(visited) is received from p_j do
(10) if (neighbors_i ⊆ visited)
(11)    then if (parent_i = i)
(12)            then the traversal is terminated        % global termination %
(13)            else send BACK(visited) to p_parent_i   % local termination %
(14)         end if
(15)    else let k ∈ neighbors_i \ visited;
(16)         send GO() to p_k; children_i ← children_i ∪ {k}
(17) end if.
```

Fig. 1.17 Time and message optimal depth-first traversal (code for p_i)

Figura 5. Algoritmo DFS (fuente original)