Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Práctica 03

Recorridos

Profesores:

Fernando Michel Tavera
Luis Mario Escobar Rosales
Brenda Ayala Flores
David Ortega Medina

Fecha de entrega

Martes , 30 de septiembre del 2025.

Lineamientos

Los alumnxs deberán hacer y entregar las prácticas siguiendo todos los puntos mencionados,si alguno llegase a ser incumplido la penalización puede variar desde puntos menos a su calificación hasta la nulificación total de éstas.

1. Las prácticas deben ser realizadas en parejas , no se calificarán prácticas individuales, así como prácticas de equipos con más de 2 miembros.
2. Las prácticas se deberán entregar a través de classroom, en un archivo .zip.Solo un miembro del equipo debe entregar la tarea con los archivos, pero ambos deben marcar como entregado.
3. El nombre del archivo .zip, debe empezar con el número de la práctica, seguido de los nombres de los integrantes.

   Practica1_FernandoMichel_DavidOrtega.zip

4. Se deberán usar únicamente las bibliotecas permitidas para dicha práctica.Queda prohibido el uso de cualquier biblioteca externa en el código.
5. Se deberá realizar un reporte en un archivo pdf que debe llevar lo siguiente :
   - Una descripción general de como se desarrolló la práctica.
   - Se debe hacer un análisis detallado en como se implementaron los algoritmos solicitados.Mencionando todas las consideraciones, etc.
   - Cualquier otro comentario o aclaración que consideren pertinente.

   Este pdf debe ir al mismo nivel que la carpeta principal de sus programas.
6. Se debe agregar un README.txt que contenga :
   - Número de la práctica
   - Nombre y número de cuenta de los integrantes

   Debe ir al mismo nivel(en la jerarquía de carpetas que el reporte).
7. Si ni el README ni el reporte llevan los nombres de los integrantes , habrá una penalización de la calificación con un punto menos.
8. Queda estrictamente prohibido el uso de cualquier código generado por Inteligencia Artificial, así como código copiado de Internet y copias entre equipos.De haber sospecha por alguna de estas situaciones, los integrantes del equipo deberán tener una entrevista con el ayudante de laboratorio.En esta entrevistas se les cuestionara aspectos de su implementaciones, algoritmos y código en general. En caso de incumplirse esta norma, la calificación de dicha práctica será automáticamente 0.
9. Si se tiene alguna complicación con la fecha y horario de entrega de la práctica , se debe avisar al ayudante para buscar una solución.
10. Si se entrega código que no compile y/o muera nada más ejecutarlo, la calificación será 0.

Computación distribuida - Recorridos

Nota : En su implementación, no deberán importar bibliotecas o usar otros archivos a menos que la práctica o el profesor lo indique.Para esta práctica bastará con usar Simpy , time y pytest.

En esta práctica implementaremos algoritmos de búsqueda,ordenamiento, etc..

1. Introducción
   Los árboles generadores creados por el algoritmo implementado la práctica pasada no necesariamente generan un "breath-first tree", es decir, un árbol donde el hijo del nodo raíz son sus vecinos , y de forma más general los procesos a la distancia d de la raíz en el árbol son los procesos a la distancia d de el proceso raíz en la gráfica de comunicación.

   BFS no implica que el árbol que es construido es independiente de la ejecución.Únicamente significa que dos procesos a una distancia de la raíz en la gráfica de comunicación están a distancia en el árbol.

   Con esto nos aseguramos que construiremos árboles más eficientes para la propagación de información, (siempre que el nodo de origen sea la raíz).

   Recordatorio: La distancia entre un proceso p_i y un proceso p_j es el longitud de el camino más corto que conecta a esos dos procesos , donde la longitud de un camino es medida por el número de canales (aristas) que lo componen.

2. Implementar las interfaces de Nodo y Canal en las siguientes clases

   - NodoBFS(Algoritmo 1)
   - NodoDFS (Algoritmo 2)

   Deberán completar la función de cada clase según el tipo de Nodo siguiendo sus respectivos algoritmos:

   - BFS (Descentralizado)
   - DFS

3. Consideraciones:

   - Por convención llamaremos padre y distancia a los atributos del nodo i que almacenarán el id del padre y la distancia al nodo distinguido en el algoritmo de BFS; a su vez en el DFS las llamaremos padre y vecinos a los atributos que necesitamos. Es importante que estos valores estén correctamente asignados a estas variables, ya que de otra manera el test no te servirá.
   - En BFS y DFS al crear un nodo, por convención, haremos que la referencia al padre sea él mismo (es algo no especificado en los algoritmos).
   - En DFS debido a que el algoritmo tiene una naturaleza no determinista, haremos una pequeña modificación:en lugar de que para cada nodo el algoritmo en la elección de un nuevo vecino tome un vértice aleatorio, se elegirá el vértice vecino con el menor id. Esto es si :

     $$N_{nv}(v_i) = \{v_3, v_6, v_2, v_7, v9, v_8\ \}$$

entonces el vértice elegido para continuar la ejecución del algoritmo sería v_2.

4. Uso Para el uso de las pruebas sigue los siguientes pasos:

   - Localiza en la misma carpeta que tus códigos fuentes el archivo test.py
   - Ejecuta el siguiente comando:

```
myUser:$ pytest -q test.py
```

Algoritmos

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)

DFS

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

when GO() is received from p_j do
(3)  if (parent_i = ⊥)
(4)     then parent_i ← j; children_i ← ∅; visited_i ← {j};
(5)          if (visited_i = neighbors_i)
(6)             then send BACK(yes) to p_j
(7)             else let k ∈ neighbors_i \ visited_i; send GO() to p_k
(8)          end if
(9)     else send BACK(no) to p_j
(10) end if.

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

Fig. 1.16 Depth-first traversal of a communication graph (code for p_i)