Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida

Práctica 02

Algoritmos básicos

Profesores:

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

Fecha de entrega

Jueves, 18 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 - Algoritmos básicos

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

   - NodoGenerador(Algoritmo 1)
   - NodoConvergecast (Algoritmo 2)
   - NodoBusqueda (Visto en clase)
   - NodoSort (Visto en clase)

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

   - Para NodoBusqueda y NodoSort, deberán emplear los algoritmos de búsqueda y ordenamiento (simples) vistos en las clases teóricas.
     Además podrán suponer que la topología de la gráfica en la que se trabajara será estrella.
   - Para el algoritmo ConvergeCast , podrán suponer que se ejecuta en una topología de árbol.

2. Funciones Auxiliares

   Para la realización de los algoritmos dados se necesitarán de dos funciones auxiliares: k_way_merge() y cuadricula:

   - k_way_merge() : Dado un arreglo $A1$, que consta de K arreglos ordenados, donde cada fila está ordenada en orden ascendente, la función regresa un único arreglo ordenado que contenga todos los elementos de todos los arreglos de $A1$.
   - Cuadricula:Dado un arreglo $B1$ y un entero n, que representa la cantidad de nodos trabajadores de una gráfica, la función regresa un arreglo de arreglos donde : la longitud del arreglo que contiene a los segmentos es igual a la cantidad de nodos trabajadores, y los arreglos contenidos son segmentos de $B1$ tales que la longitud de un segmento es la división entera de la longitud original de $B1$ y n .Considerando lo siguiente :
     - Si $longitud(B1)/n$ no es exacta , entonces tomamos el $piso$(redondeamos al resultado entero menor).Los elementos residuales se irán agregando uno en uno en cada segmento del arreglo de salida.Esto para equilibrar la carga de trabajo entre los nodos.
     - Si $n > longitud(B1)$ entonces los elementos del arreglo de salida que no alcancen elemento serán arreglos vacíos ([]).Está función debe de usarse tanto en el ordenamiento como en la búsqueda, para dividir lo más equitativamente posible la carga de trabajo entre cada nodo trabajador.

3. 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

Árbol Generador

```
1: Initially do
2: begin:
3:     if p_s = p_i then                                   ▷ Si soy el nodo distinguido
4:         parent_i = i; expected_msg_i = |neighbors_i|
5:         for each j ∈ neighbors_i do send GO() to p_j
6:         end for
7:     else parent_i = ∅                                   ▷ Si no, solo inicializo mis variables
8:     end if
9:     children_i = ∅
10: end

11: when GO() is received from p_j 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 p_j
16:         else
17:             for each k ∈ neighbors_i \ {j} do send GO() to p_k
18:             end for
19:         end if
20:     else send BACK(∅) to p_j
21:     end if
22: end

23: when BACK(val_set) is received from p_j 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
```

Convergecast

1: Initially do
2: **begin:**
3: v_i ▷ Los valores que se enviarán
4: **if** $children_i = \emptyset$ **then** ▷ Las hojas empiezan la ejecución
5: send BACK$((i, v_i))$ to $parent_i$
6: **end if**
7: **end**

8: **when** BACK$(data)$ is received from each p_j such that $j \in children_i$ do
9: **begin:**
10: $val\_set_i = \bigcup_{j \in children_i} val\_set_j \cup \{(i, v_i)\}$
11: **if** $parent_i \neq i$ **then**
12: send BACK(val_set_i) to p_k
13: **else**
14: the root p_s can compute $f(val\_set_i)$
15: **end if**
16: **end**