Facultad de Ciencias

Computación Distribuida

PRACTICA 01

Semestre 2024 – 1

Profesor:
Luis Germán Pérez Hernández

Ayudantes:
Daniel Michel Tavera
Yael Antonio Calzada Martín

Autor

Marco Silva Huerta

Edgar Montiel Ledesma

Carlos Cortés

26 de Septiembre de 2023

Algotitmo Dijkstra Distribuido

Forma de compilar

Funcionamiento

1. Definición de Estructuras de Datos:

   // Se definen tres estructuras de datos:
   - struct vertice: Representa un vértice en el grafo con un identificador único (id) y un valor de retraso (retraso).
   - struct arista: Representa una arista (conexión) entre dos vértices con información sobre el vértice de origen (origen), el vértice de destino (destino) y el peso de la arista (peso).
   - struct grafica: Representa el grafo en sí y contiene el número de vértices (nvertices), un array de vértices (vertices) y un array de aristas (aristas).

2. Inicialización del Grafo ($inicio_grafica$):

   // Esta función inicializa el grafo con un número especificado de vértices (20 en este caso).

   // Asigna memoria dinámicamente para los arrays de vértices y aristas.

   // Inicializa los vértices con valores predeterminados, donde cada vértice tiene un identificador único y un retraso inicial de 0.

   // Crea aristas entre vértices consecutivos y asigna retrasos aleatorios a las aristas. El retraso aleatorio se genera entre 1 y 1000.

3. Impresión de la Información del Grafo ($imprimir_grafica$):

   // Esta función imprime la información del grafo en forma de tabla, mostrando el ID de cada vértice y su retraso.

4. Algoritmo de Dijkstra (dijkstra):

   // Esta función implementa el algoritmo de Dijkstra para encontrar las distancias más cortas desde un vértice fuente dado (origen) hasta todos los demás vértices en el grafo.

   // Utiliza dos arrays para almacenar la información:
   - distancias: Almacena las distancias más cortas desde el vértice fuente a cada vértice en el grafo. Inicialmente, todas las distancias se establecen en INT_MAX (infinito).
   - predecesores: Almacena los predecesores en el camino más corto desde el vértice fuente a cada vértice. Inicialmente, todos los predecesores se establecen en -1.

   // Utiliza una cola de prioridad para almacenar los vértices a explorar y, en cada iteración, selecciona el vértice con la distancia más corta.

   // Actualiza las distancias y los predecesores según las aristas y los pesos del grafo.

5. Función $dijkstra_thread$ (sin implementación):

   // Esta función es la que se ejecutará en paralelo en cada hilo para calcular Dijkstra en una porción del grafo. Sin embargo, la implementación de esta función está incompleta y deberás completarla para que realice el cálculo de Dijkstra en su porción del grafo.

6. Función Principal (main):

   // En main, se crea una instancia del grafo g con 20 vértices y se imprime su información.

   // Luego, se crea un número específico de hilos ($NUM_THREADS$) que ejecutarán la función $dijkstra_thread$ en paralelo en el mismo grafo.

   // Se espera a que todos los hilos terminen su ejecución antes de liberar la memoria asignada dinámicamente para el grafo y finalizar el programa.

Pseudocódigo del Algotitmo

1. Inicializar todas las distancias en D con un valor infinito relativo,
   ya que son desconocidas al principio, exceptuando la de a,
   qué se debe colocar en 0, pues la distancia de a a si mismo sería 0.
   C es copia de V

2. Para todo vértice i en C se establece [PI]= a.

3. Se obtiene el vértice s en C tal que no existe otro vértice w en C
   tal que (D[w] < D[s]).
   Para esto se envía un mensaje al nodo correspondiente y se regresa
   un mensaje de respuesta en donde se toma el tiempo y se le asigna a
   su distancia correspondiente. De manera concurrente el nodo destino
   realiza el mismo procedimiento para calcular su distancia a sus nodos
   vecinos que no han sido visitados.

   ```
   // En lugar de buscar el vértice con la distancia más corta
   // iterativamente, ahora se utiliza una heap para mantener una lista
   // de vértices no visitados, ordenada por la distancia más corta. Así
   // encontrar el vértice n la distancia más corta en tiempo logarítmico.
   ```

4. Se elimina de C el vértice s. El vértice u se elimina del conjunto C.

5. Para cada arista e en E de longitud l,
   que une el vértice s con algún otro vértice t en C,
   Para cada arista que sale del vértice u, se verifica si la distancia
   a través del vértice u es menor que la distancia actual del vértice t.

   ```
   - Si l + D[s] < D[t], entonces:
    // Si la distancia a través del vértice u es menor que la distancia
    // actual del vértice t, entonces se actualiza la distancia del vértice t.
   - Se establece D[t] :=  l + D[s].
    // La distancia del vértice t se establece en la suma de la distancia
    // del vértice u y el peso de la arista.
   - Se establece P[t] := s.
   // El predecesor del vértice t se establece en el vértice u.
   ```

6. Se regresa al paso 4.

```
// El algoritmo regresa al paso 4 y repite el proceso hasta que todos
// los vértices hayan sido visitados.
```

Desarrollo