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 Daniel Cortés Jimenez

26 de Septiembre de 2023

Algoritmo Dijkstra Distribuido

Forma de compilar

Se añade en el readme.txt

El comando para compilar es:

$ gcc -o Practica01EdgarLedesma_CarlosCortes_MarcoSilva
Practica01_EdgarLedesma_CarlosCortes_MarcoSilva.c -pthread

El comando para ejecutar es:

$.\Practica01_EdgarLedesma_CarlosCortes_MarcoSilva

Entrada

El programa toma dos valores como entrada inicial desde dentro del código, es decir no es necesario que el usuario ingrese algo en terminal

- *nvertices*: El número de vértices que se desean en el grafo. Esto se especifica en la función MAIN() donde se establece el valor de *nvertices* (en este caso, 20).
- *vertice origen*: El programa define un vértice de origen, es decir, desde qué vértice se calcularán las rutas más cortas hacia todos los demás vértices. Esto se establece en la variable origen en la función MAIN() (en este caso, el vértice 0).

Salida

- *Retrasos*: Para cada vértice en el grafo, el retraso desde el vértice de origen hasta ese vértice y la ruta correspondiente desde el vértice de origen hasta ese vértice.
- *Árbol Generador*: También se imprime el árbol generador mínimo resultante del algoritmo de Dijkstra.

Pseudocódigo del Algoritmo

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

El algoritmo de Dijkstra es ampliamente utilizado para encontrar la ruta más corta entre dos nodos en un grafo ponderado. En esta implementación específica, estamos abordando una versión distribuida del algoritmo de Dijkstra, donde múltiples procesos se ejecutan simultáneamente para encontrar las rutas más cortas en una red de nodos interconectados.

- Inicialización

  Primero se prepara la estructura del grafo y se selecciona un nodo de origen a partir del cual se calcularán las rutas más cortas hacia todos los demás nodos. Cada vértice se inicializa con un identificador único y una lista de aristas adyacentes. Además, se establece una semilla aleatoria para garantizar que los valores de peso de las aristas sean diferentes en cada ejecución, lo que permite simular diferentes escenarios de red.

- Ejecución del algoritmo

  La parte central del código es la implementación del algoritmo de Dijkstra distribuido. Cada vértice del grafo se ejecuta en un proceso o hilo separado. Los procesos se comunican entre sí para compartir información sobre las distancias entre los vértices. El algoritmo se ejecuta de manera iterativa, donde en cada iteración se busca el vértice con la distancia más corta que aún no ha

sido visitado. Luego, se actualizan las distancias de los vértices vecinos en función de la distancia actual y el peso de las aristas.

La comunicación entre los procesos es una parte crítica de esta implementación. Para abordar esta complejidad, se utiliza la biblioteca de hilos de C++, que permite la ejecución concurrente de múltiples procesos. Cada vértice se ejecuta en un hilo separado, y los hilos comparten información a través de variables compartidas.

- Impresión de los resultados

  Una vez que se ha ejecutado el algoritmo de Dijkstra en todos los vértices y se han calculado las rutas más cortas, se imprimen los resultados. Esto incluye los retrasos del nodo de origen a todos los demás nodos y las rutas correspondientes.

Uso de hilos La elección de utilizar hilos en lugar de procesos separados se basa en la eficiencia y la facilidad de comunicación entre procesos en un sistema operativo multiproceso. Los hilos permiten que los vértices se ejecuten de manera concurrente y compartan datos de manera más eficiente, lo que es fundamental para la implementación distribuida de este algoritmo.

Funcionamiento

- `inicio_grafica(struct grafica* g, int nvertices)`:

  Este método se encarga de inicializar el grafo con retrasos aleatorios. Recibe un puntero a una estructura `grafica` y el número de vértices `nvertices` como argumentos. Primero, asigna el número de vértices al grafo. Luego, inicializa cada vértice con valores predeterminados y genera aristas con pesos aleatorios entre 1 y 1000 que conectan los vértices.

- `encontrar_minimo(int distancias[], bool conjunto_cerrado[], int nvertices)`:

  Esta función encuentra el vértice con la distancia mínima en el conjunto de vértices no visitados. Recibe un arreglo de distancias, un arreglo booleano que indica si un vértice está en el conjunto cerrado y el número de vértices como argumentos. Devuelve el índice del vértice con la distancia mínima.

- `imprimir_ruta(int predecesores[], int nodo)`:

  Este método imprime la ruta desde el origen hasta un vértice dado. Recibe un arreglo de predecesores y el nodo de destino como argumentos. Utiliza la recursión para imprimir la ruta en el formato nodo1 → nodo2 → ... → nodoN.

- `imprimir_arbol_generador(int predecesores[], int nvertices)`:

  Este método imprime el árbol generador mínimo del grafo. Recibe un arreglo de predecesores y el número de vértices como argumentos. Muestra las aristas del árbol generador, indicando las conexiones entre los vértices.

- `dijkstra_distribuido(struct grafica* g, int origen)`:

  Este es el método principal que implementa el algoritmo de Dijkstra distribuido. Recibe un puntero a una estructura `grafica` y el nodo de origen como argumentos. Calcula las rutas más cortas desde el nodo de origen hacia todos los demás nodos en el grafo. Utiliza el arreglo de distancias,

el conjunto cerrado y los predecesores para realizar los cálculos. Al final, imprime las distancias y las rutas más cortas, así como el árbol generador mínimo del grafo.

- `main()`:

  La función principal del programa. En esta función se crea una estructura `grafica`, se establece el número de vértices y el nodo de origen. Luego, se llama a `dijkstra_distribuido` para encontrar las rutas más cortas y mostrar los resultados.