Algoritmos y complejidad

Actividad 15

Integrantes:

* Kloster, Matías Nicolas. LU 110873.
* Baschiera, Luciano Manuel. LU 111847.

Profesor:

* Fillottrani, Pablo R.

# Análisis teórico Espacio-Temporal

## Determinar si un grafo no-dirigido y pesado es conexo:

### Implementando un recorrido por niveles (BFS):

Para un grafo . , .

La implementación del algoritmo se encuentra en la clase Conexo, en la carpeta src, bajo el nombre de “conexoBFS”.

Nuestra implementación consiste en realizar un recorrido por niveles en el grafo, y luego recorrer la lista de nodos “padre” para determinar si hay más de un nodo que tenga -1 como padre, lo que indica que este grafo no es conexo.

El tiempo de ejecución de conexoBFS será:

* Θ (n) para setear todos los nodos a color blanco, y poner -1 como padre de cada nodo.
* Θ (1) para crear una nueva cola enlazada.
* Θ (n\*
* (1 para setear el color del nodo actual a gris.
* + 1 para encolar el nodo en Q.
* + n) para VisitarBFS: Para cada nodo en la cola, pregunta por los arcos adyacentes, para con los cuales obtiene el nodo en la otra punta, si es un nodo blanco, le cambia el color y el padre, y lo mete en la cola (Todo Θ(1)). Se puede probar por inducción que cada nodo es metido en la cola una sola vez, por tanto ejecuta n dequeues en total.

Entonces:

El espacio de ejecución de conexoBFS será:

* Θ(3\*n) para los arreglos de color, padre y la cola.
* Θ(n2) para los arreglos de adyacencia de los n nodos (Puede considerarse n si asumimos que reutilizamos la misma lista para cada nodo)

Entonces: o si tenemos en cuenta la consideración anterior.

### Mediante el uso de la estructura disjoint-set:

Para un grafo . , .

La implementación del algoritmo se encuentra en la clase Conexo, en la carpeta src, bajo el nombre de “conexoDS”.

Nuestra implementación consiste en crear n conjuntos, uno para cada nodo, y luego para cada arco del grafo, unir los conjuntos que contienen a los nodos del arco, hasta que haya un solo conjunto disjunto.

El tiempo de ejecución de conexoDS será analizado en función de la totalidad de las operaciones de la estructura de datos Disjoint Set:

Dado que estamos usando la estructura de datos implementada con las 2 heurísticas que se proponen en el libro Cormen (union by rank & path compression), podemos decir el tiempo total de las m operaciones será del

Dado que es una función de crecimiento suave, ya que representa el logaritmo iterado, podemos decir a entradas de gran tamaño esta función está acotada por 4,

ie  **->**

El análisis de la totalidad de las operaciones se debe a que estamos implementado esta estructura de datos con 2 Heuristicas, las cuales proveen un mecanismo que incrementa la performance en las sucesivas llamadas de algunas operaciones como por ejemplo find(x).

Para este método, el cual devuelve el elemento identificador del conjunto en el cual esta el elemento pasado por paramentro(x), se uso la heurística propuesta en el libron Cormen path compression, la cual funciona de la siguiente manera: Utilizando recursión busca al elemento identificador y va seteando como padre al elemento identificador del conjunto en los nodos que fueron visitados hasta llegar a las raíz(padre), esto implica que la primera llamada a find será en el peor caso del orden lineal ya que en el peor caso tendríamos a todos los nodos en una única foresta y llamamos al método con el nodo hoja. Pero luego de esta 1er llamada, si llamamos devuelta al método, sera del orden constante, ya que quedaron todos los nodos enlazados correctamente con su respectivo elemento identificador.

Para la unión, sin embargo, utilizamos otra heurística tambien presentada en Cormen, llamada Union by rank, la cual consiste en unir dos conjuntos, es decir unión(x,y) si y solo si el conjunto de la izquierda(y) tiene la misma(o menor) cantidad de elementos que el conjunto de la derecha(x), esto garantiza una cota superior para la cantidad de veces que se actualiza el puntero de un elemento en el conjunto que pertenece x , el cual es

Donde n representa la cardinalidad del conjunto resultado de la union

Es por esto que el análisis de tiempo de ejecución se esta estructura se hace sobre la totalidad de las operaciones y no se concentra en analizar una operación aislada

|  |  |  |  |
| --- | --- | --- | --- |
| Grafo | | BFS | Disjoint Set |
| Nodos | Arcos | ------------------------- | --------------------------- |
| 5 | 4 | 660034.0ns | 1.0802928E7ns |
| 50 | 49 | 3101214.0ns | 6138476.0ns |
| 150 | 150 | 1671008.0ns | 1.0998728E7ns |
| 333 | 54326 | 594110.0ns | 515553.0ns |
| 456 | 100000 | 400679.0ns | 232907.0ns |
| 500 | 124750 | 7106027.0ns | 2533553.0ns |

Estos datos fueron tomados desde la consola de la JVM, fueron obtenidos utilizando el método System.nanoTime() el cual devuelve el valor mas preciso del timer del sistema en nanosegundos. La metodología fue utilizarlo antes de la ejecución de las verificaciones de conectitud, almacenar ese valor y luego volver a llamar al método una vez que la verificación haya terminado. Asi, restando los valores almacenados pudimos obtener una aproximación del tiempo que demoró la operación en cuestión.