Universidad Simón Bolívar

División de Ciencias Físicas y Matemáticas

Departamento de Computación y Tecnología de la Información

Laboratorio de Estructuras y Algoritmos III

DETECCIÓN DE COMPONENTES CONEXAS:
ESTUDIO EXPERIMENTAL

Profesor:

Guillermo Palma

Estudiantes:

Simón Puyosa (18-10717)

Astrid Alvarado (18-10938)

Sartanejas, 19 de junio de 2022

Resumen

El presente estudio se realiza con el objetivo de analizar los resultados respecto al tiempo de ejecución de dos algoritmos empleados para la detección de componentes conexas de grafos no dirigidos: ComponentesConexasDFS, implementado haciendo uso del algoritmo de recorrido de grafos Búsqueda en Profundidad; y ComponentesConexasCD, implementado con la estructura de datos Conjuntos Disjuntos. Ambos algoritmos fueron implementados por los estudiantes Simón Puyosa y Astrid Alvarado en el lenguaje de programación Kotlin siguiendo el pseudocódigo provisto por el libro Cormen, T. Leiserson, C. Rivest, R. and Stein, C. Introduction to algorithms, 3rd ed. MIT press, 2009.

Resultados Experimentales

A continuación, se detalla las especificaciones del computador donde se ejecutaron los algoritmos:

- Sistema operativo: Linux Ubuntu versión 20.04.2 LTS 64 bits
- Procesador: Intel i3-4005u
    a) Arquitectura: 64 bits
    b) Núcleos: 2
    c) Hilos de procesamiento: 2
    d) Velocidad mínima de procesador: 800 MHz
    e) Velocidad máxima de procesador 1.7 GHz
- Memoria RAM: 2 módulos de RAM ddr3 a 1600 MHz, 1 módulo de 4Gb y 1 módulo de 2Gb
- Versión de Kotlin: 1.5.10
- Versión de JVM: OpenJDK 11.0.11

Ahora bien, antes de proceder a analizar los resultados de las pruebas experimentales realizadas, es necesario explicar de manera breve el funcionamiento de ambos algoritmos y ciertas variaciones que fueron agregadas al momento de realizar la implementación de la misma. Por lo que se tiene lo siguiente:

Detalles para Componentes Conexas DFS:

Este algoritmo se ejecuta mediante el uso del algoritmo Búsqueda de Profundidad, también conocido como DFS (Depth-First Search), para realizar recorridos por los vértices en grafos. Para conseguir las componentes conexas, se agrega una variable global al algoritimo donde se vaya contando la cantidad de componentes conexas que el grafo encuentre cada vez que se realiza una llamada recursiva a la función dfsVisit(), finalmente, esa valor es asignado en el vértice. Para esta implementación, se realizó de forma similar al pseudocódigo expuesto "Introduction to algorithms".

Detalles para Componentes Conexas CD:

Este algoritmo es implementado mediante el uso de la estructura de datos Conjuntos Disjuntos, representados como árboles. Para efectos de implementación, se cuenta con un arreglo, el cual es inicializado de tamaño n en donde se contiene los n-1 elementos del tipo vértice que cuentan con las propiedades "padre" y "rank", necesarios para simular la representación de árboles para estos conjuntos, por otro lados, se debe tener en cuenta que la realización de dicho arreglo equivale a ejecutar la función propia de la estructura "Make-set()" n veces, por lo que el arreglo en primera instancia contendrá n conjuntos disjuntos. Además, se añade dos arreglos dinámicos auxiliares con el fin de poder almacenar información referente a los conjuntos, como el representante de cada conjunto generado y la cantidad de elementos que contiene cada uno de éstos.

Prueba Experimental

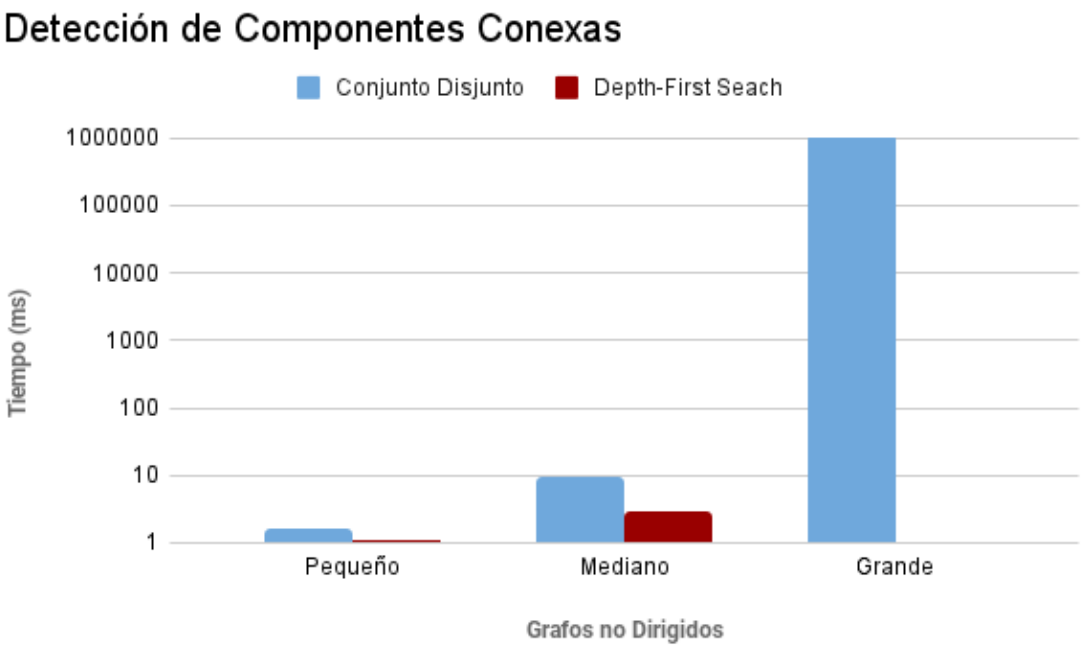
Para la obtención de los resultados experimentales, se realizó pruebas a ambos algoritmos con 3 tipos de Grafos no Dirigidos representados en un archivo de texto, a saber:

- Pequeño: grafo compuesto por 13 vértices y 13 lados.
- Mediano: grafo compuesto por 250 vértices y 1273 lados.
- Grande: grafo compuesto por 1.000.000 vértices y 7586063 lados.

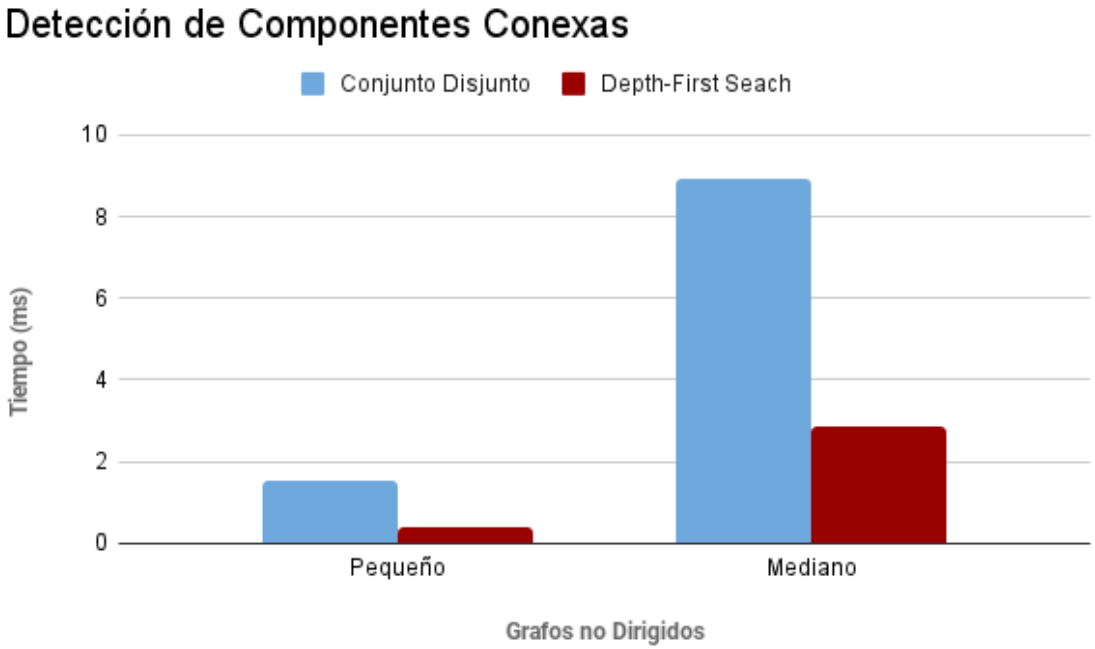
Siendo ejecutados 3 veces por cada grafo.

Gráficas:

Gráfica I: detección de componentes conexas para todos los grafos con ambos códigos.



Gráfica II: detección de componentes conexas para los grafos Pequeño y Mediano.



Datos:

Número de componentes conexas obtenidas			
	Pequeño	Mediano	Grande
ComponentesConexasDFS	3	1	—
ComponentesConexasCD	3	1	1

Tiempo de ejecución de los algoritmos			
	Pequeño	Mediano	Grande
ComponentesConexasDFS	0,3633ms	2,7946ms	—
ComponentesConexasCD	1,4863ms	8,8602ms	1080305ms

Análisis de Resultados

Al realizar ambos algoritmos, se puede concluir que implementar la detección de componentes conexas utilizando el algoritmo de Búsqueda en Profundidad se ejecuta en menor tiempo que implementar el mismo utilizando la estructura de datos Conjuntos Disjuntos, lo cual corresponde con el tiempo esperado visto en clase de dichos algoritmos, O(|V|+|E|) y O(|E|$\propto^{-1}$(|V|)) respectivamente. Se debe recordar la función inversa de Ackermann $\propto^{-1}$ (x) tiende a comportarse como una constante, en particular $\propto^{-1}$ (x)≤4, sin embargo cabe destacar que sigue siendo una función recursiva que crece lentamente y que por lo general tiende a ser menor o igual a 4. Dicho esto, dado que los grafos de prueba incluyen números de lados muy grandes, obtener como resultado que la detección de componentes conexas usando el algoritmo de DFS se ejecutase mucho más rápido que la implementación con Conjuntos Disjuntos es un resultado esperado.

Por otro lado, se puede observar cierta desventaja al momento de implementar este algoritmo de detección con DFS, y es que como este algoritmo de recorrido de grafos es un algoritmo recursivo, las llamadas a ésta se almacenan en memoria y al ejecutarla con números grandes ocasiona que la memoria se llene y en consecuencia ya no se pueda retener esa información, obteniendo el error Stack Overflow. Por lo que, para números muy grandes, como

viene siendo el caso del grafo Grande, no es posible realizar la detección de componentes conexas con lo cual es necesaria una mejor implementación del mismo, en cambio, utilizando la implementación de Conjuntos Disjuntos, a pesar de ser más tardado, se logra ejecutar correctamente el algoritmo retornando el resultado esperado.

En conclusión, detectar las componentes conexas empleando DFS se ejecuta en tiempo menor a realizarlo con la estructura de datos Conjuntos Disjuntos, sin embargo, para números muy grandes, utilizar la implementación con Conjuntos Disjuntos hace que se realice correctamente la detección de componentes conexas, pero de forma ineficiente por lo tardado del mismo.