Practica de Métodos Algorítmicos de Resolución de Problemas

Pablo Vazquez Gomis
Universidad Complutense de Madrid

January 3, 2019

Abstract

Practica opcional del primer cuatrimestre para la asignatura de Métodos Algorítmicos de Resolución de Problemas.

1 Introducción

1.1 Descripción de la práctica

Implementar o Java o en C++ un algoritmo, que dado un grafo dirigido, detecte si tiene o no ciclos. En caso de ser acíclico, ha de listar sus vértices en orden topológico. Si hay más de uno posible, los puede listar en cualquiera de ellos. En caso de ser cíclico, ha de listar sus componentes fuertemente conexas (cada una es un conjunto de vértices). El algoritmo para esta segunda parte puede verse en el Capítulo 22 del Cormen (2001, segunda edición).

1.2 Implementación

La práctica se puede dividir en 3 partes diferenciales:

1. Comprobar si un grafo es cíclico o no
2. los vértices en orden topológico
3. Listar los componentes fuertemente conexos

Dado que en al ordenar los vértices de el punto 2 uno de los casos base devuelve error si el grafo es cíclico y su complejidad esta en orden $O(|V| + |E|)$ utilizaremos esa función para comprobar el punto 1.

En caso positivo, habremos encontrado una solución al punto 1 en orden $O(|V| + |E|)$. En caso negativo, tendremos la solución al punto 2 en el mismo orden.

Para resolver el punto 3 utilizamos el algoritmo de Tarjan[1] para encontrar componentes fuertemente conexas, el cual resuelve el problema utilizando *Depth First Search (DFS)* con una complejidad de $O(|V| + |E|)$.

2 Ordenación topológica

Definición: Usando la definición propuesta por A. B. Kahn[2]:

> A list in topological order is such that no element appears in it until after all elements appearing on all paths leading to the particular element have been listed.

En otras palabras, dado que es necesario que el grafo sea acíclico para poder ordenarlo topológicamente, el raíz del grafo sera el primer elemento de la lista ordenada y los elementos siguientes serán la ordenación topológica de sus hijos.

Implementación: Usando DFS, podemos recorrer la lista desde una raíz del grafo cualquiera hasta el final de un camino, Continuando esta estrategia, la lista se compondrá recursivamente:

$$vertice + dfsTopologicalSort(e) \mid \forall e \in node.edges$$

Esto se consigue, añadiendo el nodo a la lista en primera posición cuando todas su aristas han sido visitadas.

3 Componentes fuertemente conexos

Definición: Dado un grafo dirigido G, supongamos que para cada par de vértices v,w existe los caminos: $p_1 : v \Rightarrow w$ y $p_2 : w \Rightarrow v$, entonces G es fuertemente conexo.

Implementación: El problema lo resolveremos con el algoritmo propuesto por Robert E. Tarjan[1], el cual se basa en estos puntos:

1. Usando *DFS* busca por todos los nodos siguiendo los caminos del grafo, asignando un indice y un **lowpoint-index** único a cada uno la primera vez que se visita el nodo.
2. El **lowpoint-index** representa el índice del nodo mas bajo al que se puede llegar siguiendo los caminos desde el nodo inicial y se va actualizando en el backtranking de *DFS*.
3. Cuando el siguiendo un camino, se llega a el nodo inicial, todos los nodos que tengan el mismo **lowpoint-index** son nodos pertenecientes a el componente fuertemente conexos

4. Para evitar que el algoritmo dependa de el nodo inicial, que se elige aleatoriamente, se lleva la cuenta de los nodos recorridos con un stack. A la hora de recuperar componentes fuertemente conexas, solo los nodos que están en el stack pertenecen a él.

4 Ejecuciones Sencillas

Listing 1: resources/graph1.txt

```
Cycle number 0
0
Cycle number 1
4
Cycle number 2
3 -> 1 -> 2
Time to complete tarjan: 0.027
Time to complete cycles search: 0
```

Listing 2: resources/graph2.txt

```
1 - 2 - 3 - 4 - 5 - 6
Time to complete topological sort: 0.039
Time to complete cycles search: 0.009
```

Listing 3: resources/graph3.txt

```
Cycle number 0
0 -> 1 -> 2
Cycle number 1
6 -> 4 -> 5
Cycle number 2
7 -> 3
Time to complete tarjan: 0.187
Time to complete cycles search: 0.012
```

5 Gráficas

Todas las gráficas se encuentran en el apéndice de este documento.

5.1 Observaciones

Como se puede observar en las gráficas, el algoritmo de Tarjan[1] crece de forma mas rápida el algoritmo de para ordenar topológicamente la lista, esto aparenta ser debido a el coste acumulado de las operaciones sobre vectores y listas.

Esta conclusión es inferida de la comparación entre la búsqueda en de ciclos y la ordenación topológica. Ambos hacen esencialmente la misma búsqueda en profundidad, salvo por una pequeña operación de almacenaje en el stack pero sin embargo, como podemos observar en la figura uno, la búsqueda topológica crece de manera mas veloz que el algoritmo de búsqueda.

Con el algoritmo de Tarjan[1] ocurre exactamente lo mismo pero aumentado dado que además de mantener el stack de llamadas, hemos de ir añadiendo a la lista de componentes.

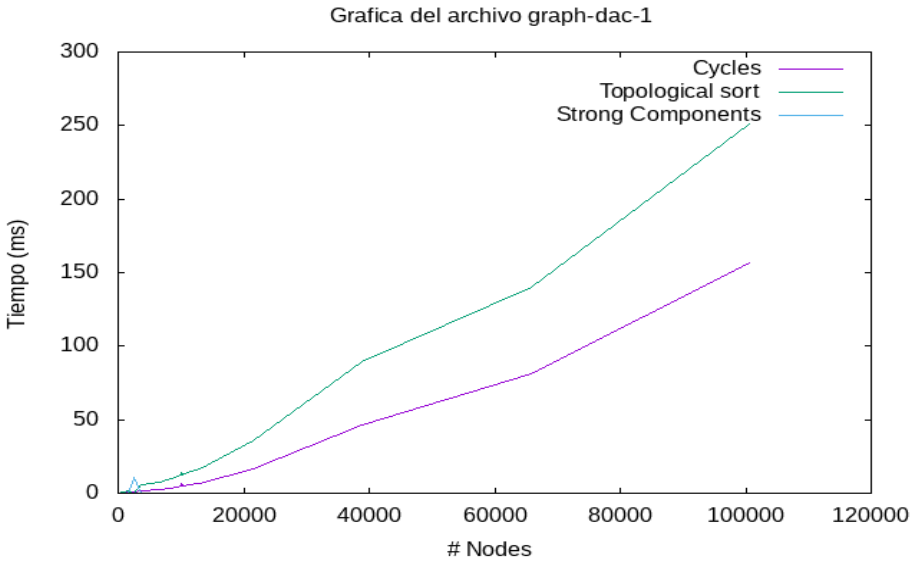
Conclusión Aunque las operaciones con las estructuras de datos no añaden a la complejidad del algoritmo, si que se ven reflejadas en los tiempos de ejecución.

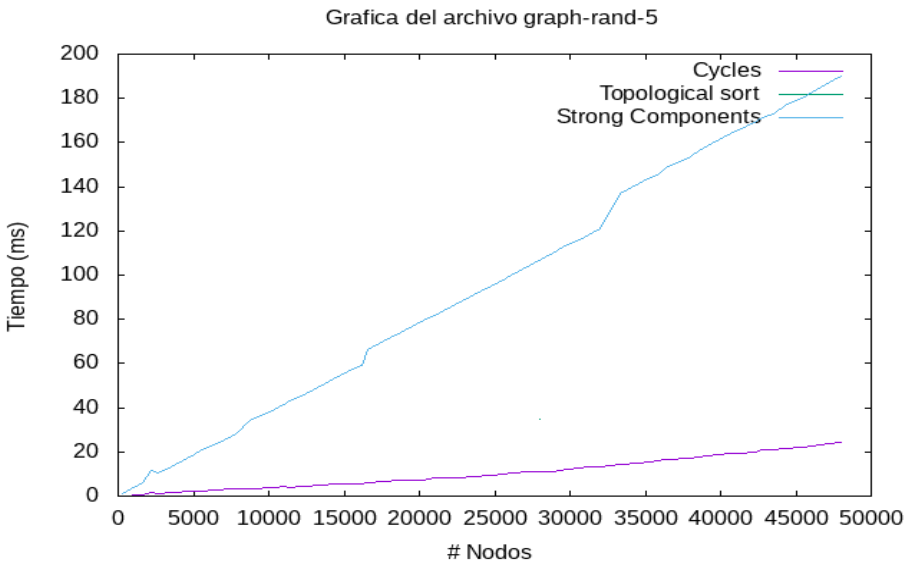
References

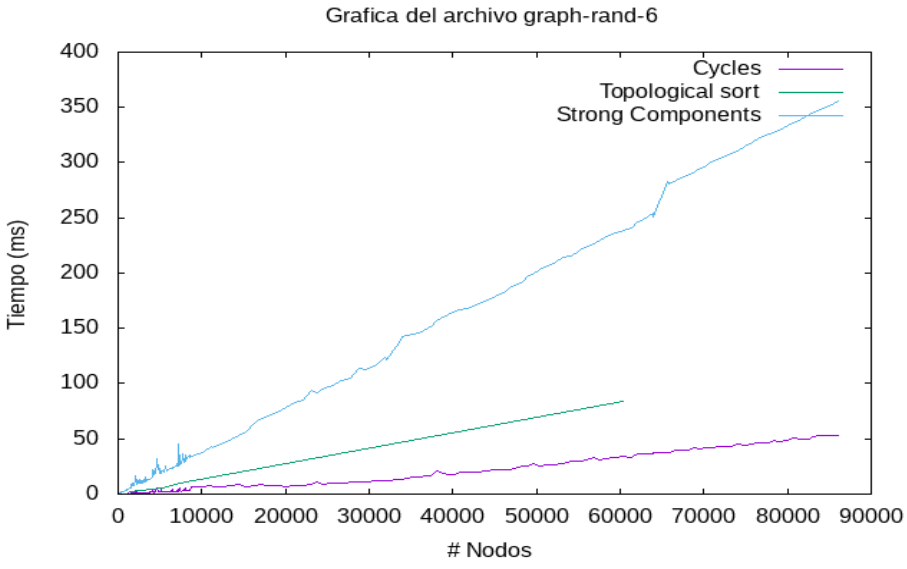
[1] Robert E. Tarjan. *Depth-First Search and Linear Graph Algorithms*. Stanford University. Stanford, California.

[2] A. B. Kahn. *Topological sorting of large networks*. Westinghouse Electric Corporation, Baltimore, Maryland

6 Apendice




Grafica del archivo graph-rand-5
Cycles
Topological sort
Strong Components
Tiempo (ms)
Nodos
0
20
40
60
80
100
120
140
160
180
200
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000


Grafica del archivo graph-rand-6
Cycles
Topological sort
Strong Components
Tiempo (ms)
Nodos
0
50
100
150
200
250
300
350
400
0
10000
20000
30000
40000
50000
60000
70000
80000
90000