Practica de Metodos Algoritmicos de Resolución de Problemas

Pablo Vazquez Gomis
Universidad Complutense de Madrid

December 29, 2018

Abstract

Practica opcional del primer cuatrimestre para la asignatura de Metodos 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 diferenciables:

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 funcion 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 seran la ordenacion topológica de sus hijos.

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

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

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

3 Componentes fuertemente conexos

Definición: Dado un grafo dirigido G, supongamos que para cada par de vertices 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** unico a cada uno la primera vez que se visita el nodo.
2. El **lowpoint-index** representa el indice 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 pertencientes 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 estan en el stack pertenecen a él.

4 Conclusions

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