Práctica 9: Algoritmos sobre grafos

Ejercitación básica

Ejercicio 1. Diseñar un algoritmo que reciba como entrada un grafo $G = (V, E)$ y determine si es *conexo*, es decir, si para todo par de vértices $v, w \in V$ existe un camino de v a w.

Comentario: una manera ingenua de resolverlo es recorrer todos los pares de vértices $v, w \in V$ y ver si existe un camino de v a w usando un algoritmo de alcanzabilidad, lo que tiene complejidad temporal en peor caso $O(n^2 \cdot (n + m))$. Pensar en un algoritmo que mejore la complejidad. Es posible resolverlo en $O(n + m)$.

Ejercicio 2. Diseñar un algoritmo que reciba como entrada un grafo $G = (V, E)$ y determine si es *acíclico*, es decir, si no tiene ningún ciclo. Recordemos que un ciclo es un camino de longitud $\ell > 0$ que va de algún vértice $v \in V$ a sí mismo. Es posible resolverlo con complejidad temporal $O(n + m)$ en peor caso.

Ejercicio 3. Sea $G = (V, E)$ un grafo. Recordemos que el problema de alcanzabilidad recibe como entrada dos vértices $v, w \in V$ y devuelve como salida un booleano que indica si hay un camino de v a w en G.

Modificar el algoritmo de alcanzabilidad visto en clase (basado en DFS) para que además de indicar si hay un camino de v a w devuelva la lista de todos los vértices intermedios que llevan de v a w en el camino encontrado.

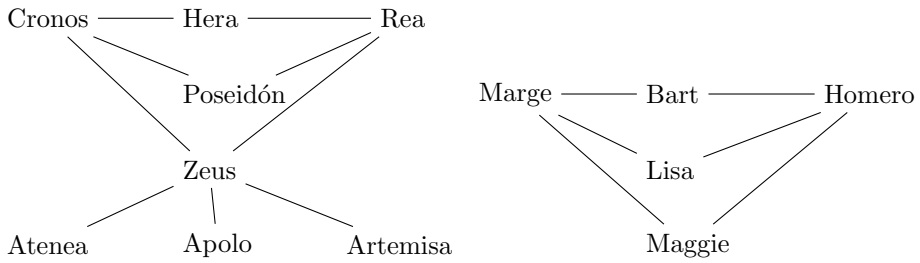
Ejercitación adicional

Ejercicio 4. Recordemos que, en un grafo $G = (V, E)$, llamamos *distancia* de v a w a la longitud del camino más corto que conecta a v con w. La distancia de un vértice a sí mismo es siempre 0. Exhibir un grafo de ejemplo y dos vértices v, w de tal modo que el algoritmo de alcanzabilidad basado en DFS encuentre un camino de v a w pero que no sea el más corto.

Ejercicio 5. Como continuación del ejercicio anterior, adaptar el algoritmo de alcanzabilidad basado en BFS para que dado un grafo $G = (V, E)$ y dos vértices $v, w \in V$ determine la distancia de v a w.

Ejercicio 6. En un grafo $G = (V, E)$ una *componente conexa* es un conjunto de vértices que están todos conectados entre sí por caminos, sin excluir a ningún vértice conectado.

Por ejemplo, un grafo $G = (V, E)$ representa relaciones de parentesco. Cada vértice representa una persona. Dados dos vértices $v, w \in V$, hay una arista $\{v, w\} \in E$ si v es el progenitor y w es uno de sus hijos, o viceversa. Dos personas son *parientes* si hay un camino de v a w. Las componentes conexas del grafo se corresponden con *familias*, es decir, conjuntos de vértices que contienen a grupos de personas que son todos parientes entre sí, sin excluir a ningún pariente. Por ejemplo, el siguiente grafo tiene 2 familias:



Proponer un algoritmo que dado un grafo determine la cantidad de familias.