uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Tema 6 - Grafos

Hoja de Problemas

Problema 1 -

a) En la clase Graph_Matrix, implementación de un grafo basado en matriz, (fichero graph_any_type_matrix.py), añade un método get_adjacentes que reciba un vértice, vertex, y que devuelva una lista de Python conteniendo los vértices adyacentes a vertex. La lista únicamente contiene los vértices, no los pesos.

b) Implementa además un método, get_origins, que reciba un vértice, vertex, y que devuelva una lista de Python conteniendo aquellos vértices que sean origen de alguna arista cuyo destino sea vertex. La lista únicamente contiene los vértices, no los pesos.

Solución: ver graph_any_type_matrix.py

Problema 2 -

c) En la clase Graph, implementación de un grafo basado en diccionario, (fichero graph.py), añade un método get_adjacentes que reciba un vértice, vertex, y que devuelva una lista de Python conteniendo los vértices adyacentes a vertex. La lista únicamente contiene los vértices, no los pesos.

d) Implementa además un método, get_origins, que reciba un vértice, vertex, y que devuelva una lista de Python conteniendo aquellos vértices que sean origen de alguna arista cuyo destino sea vertex. La lista únicamente contiene los vértices, no los pesos.

Solución: ver graph.py

Problema 3:

Sea Graph2 una clase hija de Graph (implementación de grafo basada en diccionario). Implementa una **versión iterativa del método dfs** para obtener el recorrido en profundidad de un grafo. Pista: Se recomienda usar una estructura de pila para almacenar los nodos que se van visitando en cada camino.

Nota: La solución está incluida en el fichero traversals.py

Problema 4:

En la clase Graph3 (fichero minimum_path.py), hija de la clase Graph (implementación de un grafo basado en diccionario), implementa un método, **minimum_path**, que reciba dos vértices, start y end, y devuelva una lista que contenga el camino mínimo de start a end, y la distancia del camino. Si el camino no existe, deberá devolver una lista vacía y distancia infinito.

Solución: ver la función minimum_path en minimum_path.py

Problema 5:

Dado un un **grafo dirigido no ponderado**. Implementa la función ***minimum_path***, que reciba dos vértices, *start* y *end*, y devuelve una lista de Python con los vértices que forman el camino mínimo desde start a end, ambos inclusive. La función minimum_path debe utilizar el algoritmo de camino mínimo de Dijkstra. Como el grafo es un grafo no ponderado, se considera como camino mínimo aquel que tenga un menor número de aristas. Está permitido utilizar estructuras de Python como las listas o los diccionarios.

Solución: ver la función minimum_path en minimum_path.py

Problema 6:

Implementa un método, **non_accesible**, que reciba un vértice, vertex, y devuelva la lista vértices del grafo que no son accesibles desde vertex. Un vértice end es no accesible desde vertex si no existe ningún camino de vertex a end.

Solución: ver la función non_accesible en minimum_path.py

Problema 7:

Implementa un método, **get_recheable**, que reciba un vértice, vertex, y devuelva la lista todos los vértices para los que existe un camino desde vertex.

a) Implementa una solución basada en bfs.

b) Implementa una solución basada en dfs.

Solución: ver la función get_reachable en problems_graph.py

Problema 8:

Implementa un método, **has_cycle**, que comprueba si un grafo no dirigido contiene un grafo.

a) Implementa una solución que utilice el algoritmo dfs (búsqueda en profundida).
b) Implementa una solución que utilice el algoritmo bfs (busqueda en amplitud).

Solución: ver la función has_cycle en problems_graph.py

Problema 9:

Implementa un método, **has_cycle**, que comprueba si un grafo dirigido contiene un grafo. La solución debe estar basada en el algoritmo dfs (búsqueda en profundida).

Solución: ver la función has_cycle en problems_graph.py

Problema 10:

Implementa un método, **check_bipartite**, que comprueba si un grafo es bipartito. Un **grafo bipartito** es un **grafo** tal que su conjunto de vértices puede particionarse en dos conjuntos independientes.

Solución: ver la función check_bipartite en problems_graph.py