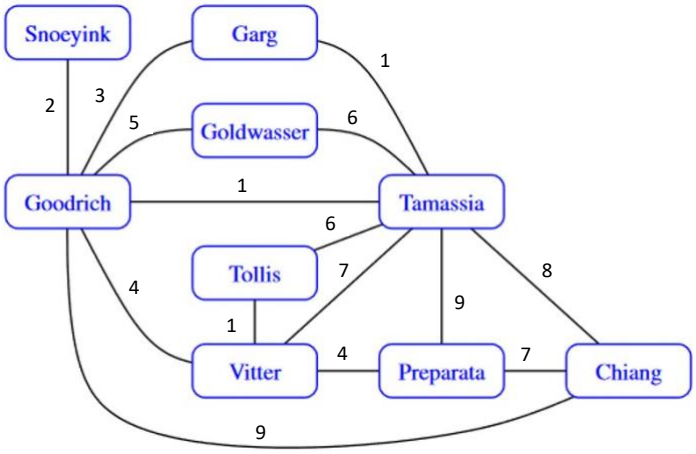
Práctica 5. Implementación de grafos y algoritmos sobre grafos

Ejercicio 0: Hacer uso de la implementación vista en clase de grafo para representar el que se incluye a continuación:



Ahora, realiza las siguientes tareas:

1. Añade a la implementación del grafo disponible en el fichero *graph.py* un método para poder devolver una cadena de caracteres de un grafo, y prueba su funcionamiento imprimiendo el grafo representado arriba. La cadena a devolver debe tener la estructura vista en el ejemplo estudiado en clase de leer un grafo de un fichero.
2. Imprime el número de vértices y aristas del grafo.
3. Imprime el grado del vértice “Tamassia”.
4. Imprime las aristas que inciden en el vértice “Tollis”.

Ejercicio 1: Dados los siguientes casos indicar si se debería usar una lista de adyacencia, un mapa de adyacencia o una matriz de adyacencia:

1. Para representar un grafo de 10000 vértices y 20000 aristas, siendo importante usar el menor espacio posible.
2. Necesitamos la respuesta a get_edge(u,v) tan rápido como sea posible y sin importar el espacio usado.

Ejercicio 2: Sea un grafo no dirigido G, cuyos vértices son enteros del 1 al 8 y el conjunto de vértices adyacentes de cada vértice es el siguiente:

Vértice	Vértices adyacentes
1	(2,3,4)
2	(1,3,4)
3	(1,2,4)
4	(1,2,3,6)
5	(6,7,8)
6	(4,5,7)
7	(5,6,8)
8	(5,7)

Asumiendo que, en un recorrido de G, los vértices adyacentes de un vértice dado son devueltos en el mismo orden de la lista anterior:

1. Dibujar G.

2. Dar la secuencia de vértices de G visitados usando un recorrido DFS empezando en el vértice 1.
3. Dar la secuencia de vértices de G visitados usando un recorrido BFS empezando en el vértice 1.

Ejercicio 3: Desarrollar una función que imprima un grafo en un fichero, siguiendo el formato visto en clase para el programa que lee un grafo de un fichero.

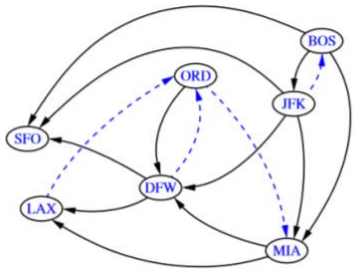
Ejercicio 4: Resolver el ejercicio 2 usando una implementación de los recorridos DFS y BFS. Comparar los resultados. Después añade un enlace entre los vértices 2 y 5 y vuelve a realizar los recorridos, justificando el resultado obtenido.

Ejercicio 5: Desarrollar una función para detectar si hay un ciclo en un grafo no dirigido.

Ejercicio 6: Desarrollar una función para detectar si hay un ciclo en un grafo dirigido.

Ejercicio 7: Desarrollar una función que imprima en pantalla el árbol de expansión mínimo generado por el Algoritmo de Prim.

Ejercicio 8: Calcular el orden topológico del grafo dirigido siguiente (tener en cuenta las líneas sólidas):



Ejercicio 9: A Bob le encanta estudiar lenguas extranjeras y quiere planificar sus estudios para los próximos años. Está interesado en los cursos de 9 lenguas: L15, L16, L22, L31, L32, L126, L127, L141 y L169. Quiere hacerlos todos, pero existen requisitos previos de haber cursado otros cursos con antelación. Los prerrequisitos de los cursos son los siguientes:

- L15: Ninguno
- L16: L15
- L22: Ninguno
- L31: L15
- L32: L16, L31
- L126: L22, L32
- L127: L16
- L141: L22, L16
- L169: L32

¿En qué orden debe cursar Bob estos cursos respetando los requisitos?

Ejercicio 10: Hay 8 pequeñas islas en un lago y se quiere construir 7 puentes para conectarlas de forma que cada isla pueda ser alcanzada desde cualquier otra a través de los puentes. El coste de un puente es proporcional a su longitud. Las distancias entre pares de islas son las siguientes:

	1	2	3	4	5	6	7	8

1		240	210	340	280	200	345	120
2			265	175	215	180	185	155
3				260	115	350	295	230
4					160	330	295	230
5						360	400	170
6							175	205
7								305
8								

Encontrar dónde se deben situar los puentes para minimizar el coste de la construcción.

Ejercicio 11: Implementa un sistema de cálculo de rutas. Para ello, realiza las siguientes tareas:

1. Crea un grafo las principales calles de la ciudad de Cartagena o Murcia, partiendo de un mapa digital on-line (por ejemplo, Open Street Maps), identificando una serie de cruces que serán destinos de interés. Considera alrededor de 20 destinos para el grafo, y representa las distintas conexiones mediante la distancia en metros.
2. Representa el grafo en un formato de fichero que permita ser leído con la función vista en clase de lectura de grafos desde ficheros.
3. Implementa un programa que cargue el grafo del punto anterior y que solicite por pantalla el origen y el destino de la ruta.
4. Usa la implementación del algoritmo de Dijkstra para calcular la distancia más corta desde el nodo de origen a todos los demás nodos, y reconstruye la ruta más corta desde el nodo de origen al destino.
5. Muestra el resultado por pantalla, indicando los distintos cruces (o puntos de interés) por los que habría que pasar para recorrer menor distancia desde el origen al destino.