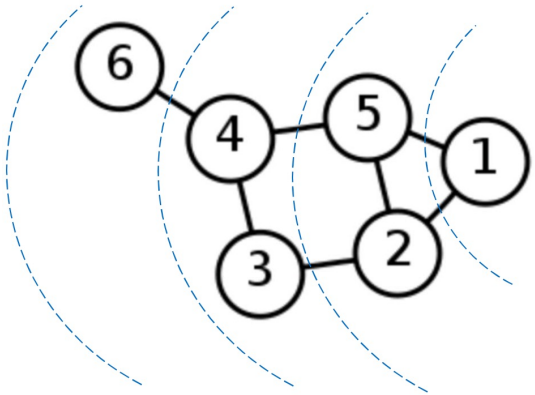
Algoritmos y Programación

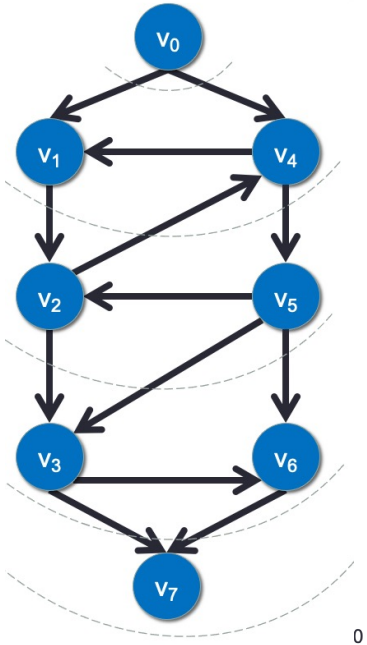
Práctica 3: Distancia Mínima (en número de saltos)

Distancia Mínima (en número de saltos) en un grafo no-dirigido

- BFS



- Shortest Path Length



Recorrido: V_0 V_1 V_4 V_2 V_5 V_3 V_6 V_7

Distancia

0	1	2	3	1	2	3	4
V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7

Tienes la descripción completa del algoritmo que debes programar en la clase de teoría.

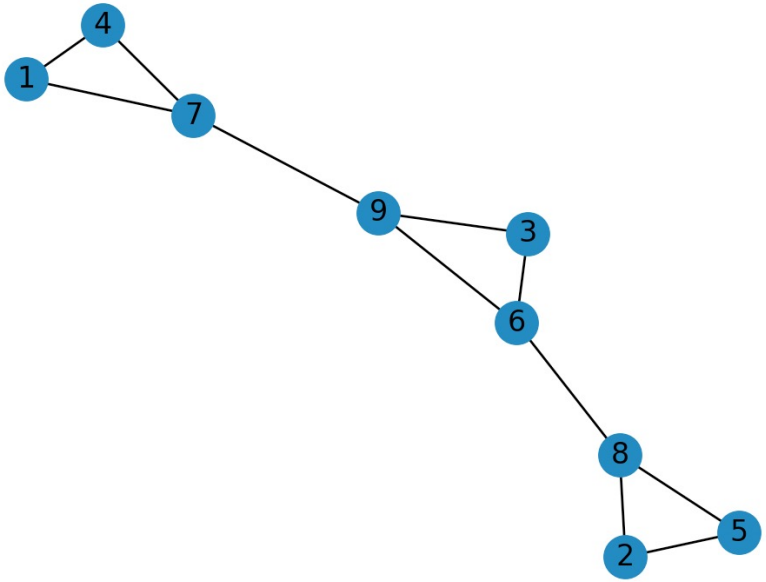
Formato de los datos de entrada

El mismo formato del grafo no-dirigido de la práctica anterior:

- La primera línea es un descriptor: número de vértices, número de aristas.
- El resto de las líneas son las aristas.

Ejemplo:

```
9 11
1 4
2 8
3 6
4 7
5 2
6 9
7 1
8 5
8 6
9 7
9 3
```



Pasos de esta práctica

1. Añade al fichero *graph_utils.py* tu código del ejercicio de la semana anterior para crear un grafo no-dirigido.

2. Programa en el fichero *solve.py* el algoritmo “Shortest Path Length” explicado en clase de teoría para calcular la distancia minima (en número de saltos) desde el primer vértice del grafo al resto de los vertices.

Fíjate que aunque el ejemplo que vimos en clase calculaba la distancia en un grafo dirigido, podemos utilizar este algoritmo en grafos no-dirigidos (que es lo que vas a programar en esta práctica).

VPL

graph_utils.py

```python
import networkx as nx

def build_graph():
    # Añade aqui el código que hiciste en el ejercicio de la
    # semana anterior para crear un grafo no-dirigido.

    return graph

```

simple_queue.py

```python
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)
```

solve.py

```python
import networkx   as nx
from sys          import maxsize as infinite
from simple_queue import *

def bfs_path_length(graph, first_node):
    """
    Compute the shortest path length of the non-directed graph G
    starting from node first_node. Return a dictionary with the
    distance (in number of steps) from first_node to all the nodes.
    """

    distance = {}                 # Diccionario con la distancia desde
                                  # firstNode al resto de los nodos.
    for node in graph.nodes():
        distance[node] = infinite

    # solve it here!
    # ...

    return distance
```