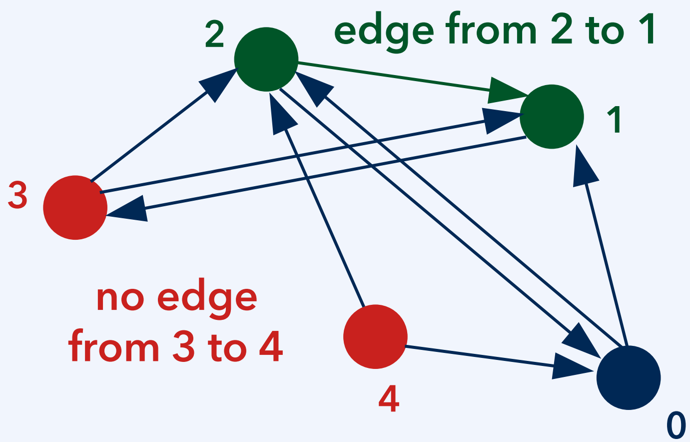
# Travelling salesman: Tutorial 3.5 problem

CO2412

8th March 2022

# Adjacency matrix data structure

**Matrix-like data structures** in Python include lists of lists (*i.e.*, 2D dynamic arrays), if the numpy library is used, two-dimensional static arrays. For graphs, the most relevant data structure of this type is the **adjacency matrix**.




```
self._adj_matrix = [  [0,  1,  1,  0,  0],
                      [0,  0,  0,  1,  0],
                      [1,  1,  0,  0,  0],
                      [0,  1,  1,  0,  0],
                      [1,  0,  1,  0,  0]  ]
```

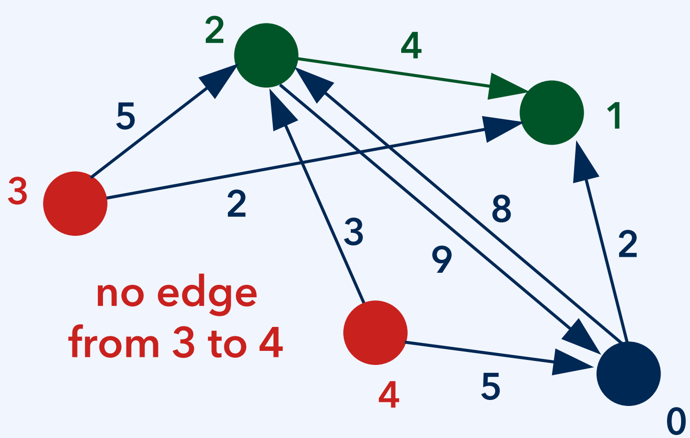
For a sparse graph, the majority of entries in the 2D array/matrix is zero. Adjacency matrices are commonly used when expecting a **dense graph**.

**self**._adj_matrix[2][1] = **1**, or **True**

**self**._adj_matrix[3][4] = **0**, or **False**

# Adjacency matrix data structure

For a graph with labelled edges, the adjacency matrix contains edge labels.
In the case of a weighted graph, the labels represent the length of the edges.
If these edges are travel distances, the diagonal entries should be zero.




```
self._adj_matrix = [ [0, 2, 8, ∞, ∞],
                     [∞, 0, ∞, ∞, ∞],
                     [9, 4, 0, ∞, ∞],
                     [∞, 2, 5, 0, ∞],
                     [5, ∞, 3, ∞, 0] ]
```

For a sparse graph, the majority of entries in the 2D array/matrix is infinity. Adjacency matrices are commonly used when expecting a **dense graph**.

**self**._adj_matrix[1][1] = **0**

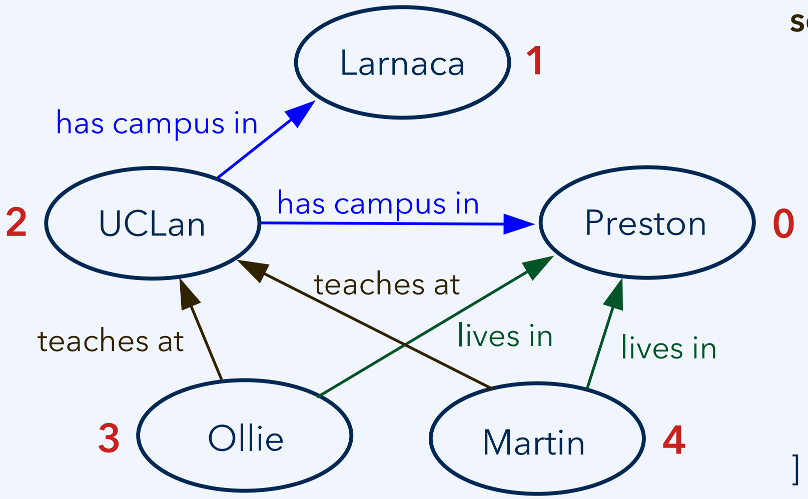
**self**._adj_matrix[2][1] = **1**

**self**._adj_matrix[3][4] = **∞**

# Adjacency matrix data structure

For a graph with labelled edges, the adjacency matrix contains edge labels.

**Task 3.5.1c: Adaptation to the assessment problem.**




```
self._adj_matrix = [
    [None,          None,   None,          None,   None],
    [None,          None,   None,          None,   None],
    ["has campus in", "has campus in", None, None, None],
    ["lives in",    None,   "teaches at",  None,   None],
    ["lives in",    None,   "teaches at",  None,   None]
]
```

```
self._node_labels = [
    "Preston",
    "Larnaca",
    "UCLan",
    "Ollie",
    "Martin"
]
```

For a sparse graph, the majority of entries in the 2D array/matrix is None. Adjacency matrices are commonly used when expecting a dense graph.

# Adjacency matrix data structure

**Task 3.5.1b: Implement calculation of the length of a path.**

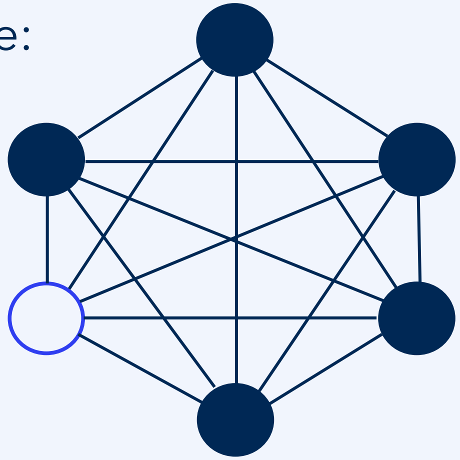
```
# returns weight of the edge between i and j
#
def get_weight(self, i, j):
    return self._adj_matrix[i][j]

# the path is a list of node IDs
#
def get_length_of_path(self, path):
    length = 0
    for i in range(len(path) - 1):
        length += self.get_weight(path[i], path[i+1])
    return length
```

**Path given by a list of traversed nodes.** The length of the path is:

```
_adj_matrix[ path[0] ][ path[1] ]
    + _adj_matrix[ path[1] ][ path[2] ]
    + _adj_matrix[ path[2] ][ path[3] ]
    ...
    + _adj_matrix[ path[len(path) - 2] ]
            ][ path[len(path) - 1] ]
```

# Travelling salesman problem (TSP)

Example:
$n = 6$



start and end point

How many cycles covering all nodes are there in a **complete graph** with $n$ nodes, that is a graph where every node is adjacent to every other node?

**Task 3.5.2a: No. of Hamilton cycles**

A travelling salesman needs to visit all the cities, by a path that ends at the same city where it starts (a **cycle**).

No city may be visited twice. Every city must be visited exactly once. (Except for returning to the start.) These cycles are **Hamilton cycles**.

Assume that the initial/final node is fixed; let it be the node no. 0.

No. of Hamilton cycles in a complete graph: $(n - 1) \cdot (n - 2) \cdot \ldots \cdot 2 \cdot 1 = (n - 1)!$

$n = 12$ nodes: $11! = 39.9$ million cycles; $n = 15$ nodes: $14! = 87.2$ billion cycles.

# TSP: Randomized approximation algorithm