This example shows two ways to represent graphs in Python: adjacency matrix and adjacency list.

The adjancency matrix

To create the adjacency matrix the least information we must know is how many nodes/vertices does the graph have. The matrix can be implemented in $N \times N$ list, where N is number of vertices.

Example 1: creating an empty adjacency matrix in Python

```
# example_1.py

N = 5

graph_matrix = [[0] * N for i in range(N)]

for i in range(N):
    for j in range(N):
        print(f"{graph_matrix[i][j]:3d}", end="")
    print()
```

The output:

```
$ python3 example_1.py
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
```

Each `graph_matrix[i][j]` stores the weight of the edge from vertex i to vertex j. This implementation is very flexible because you can use it to both directed or undirected graphs and looking up weights between vertices can be done quickly.

The adjancency list

Again the information about the graph we must know is the number of vertices. Implementing the empty adjacency list in Python does not differ much from implementing the graph as adjacency matrix:

Example 2: creating the empty adjacency list in Python

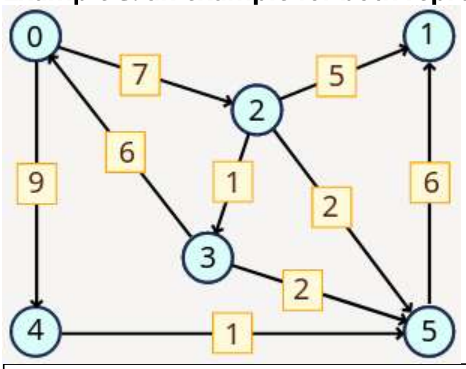
```
N = 5

graph_list =  [[] for i in range(N)]

for i in range(N):
    print(f"vertex {i} neightbours: {graph_list[i]}")
```

The output:

```
$ python3 example_2.py
vertex 0 neightbours: []
vertex 1 neightbours: []
vertex 2 neightbours: []
vertex 3 neightbours: []
vertex 4 neightbours: []
```

Now each list stores only the neighbour vertex or vertex-weight pairs. When we are looking for all neightbours of some particular vertex i using the adjacency matrix is not the best choice. For adjacency we simply return `graph_list[i]`. However, looking for particular weight between two vertices is more time consuming in this structure.

Example 3: an example for both representations in Python




```
graph_matrix = [
#    0  1  2  3  4  5
    [0, 0, 7, 0, 9, 0], # 0
    [0, 0, 0, 0, 0, 0], # 1
    [0, 5, 0, 1, 0, 2], # 2
    [6, 0, 0, 0, 0, 2], # 3
    [0, 0, 0, 0, 0, 1], # 4
    [0, 6, 0, 0, 0, 0]  # 5
]

# each pair = (neighbour, weight)
graph_list = [
    [(2, 7), (4, 9)],           # 0
    [],                         # 1
    [(1, 5), (3, 1), (5, 2)],   # 2
    [(0, 6), (5, 2)],           # 3
    [(5, 1)],                   # 4
    [(1, 6)],                   # 5
]
```

Last modified: Thursday, 9 November 2023, 5:23 PM