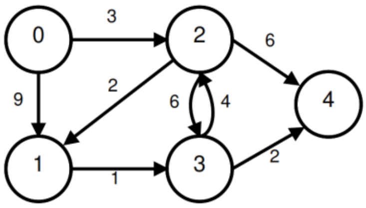
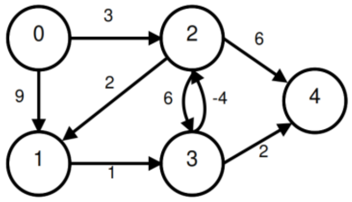
Taller 3

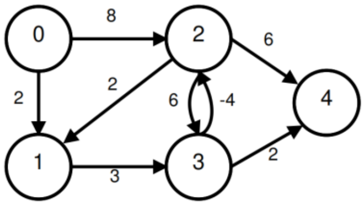
Considere los siguientes grafos:



(a)



(b)



(c)

1. Para los 3 grafos, calcule (por inspección) la longitud los caminos más cortos desde el nodo 0 hacia el resto de nodos.
    a. Grafo a:

Nodo	1	2	3	4
Distancia a 0	5	3	6	8

    b. Grafo b:

Nodo	1	2	3	4
Distancia a 0	5	3	6	8

    c. Grafo c:

Nodo	1	2	3	4
Distancia a 0	2	8	5	7

2. Para el grafo b, ¿puede encontrar un camino desde 0 a 1 de costo 0?
Respuesta: Sí, ya que siguiendo la secuencia: 0 -> 2 -> 1 -> 3 -> 2 -> 1 varias veces la longitud del camino se reduce en 1 por cada vuelta que se hace por el ciclo negativo 2->1->3->2. Como la longitud del camino sin hacer ciclos es 5 solo basta con realizar el ciclo 5 veces.

3. Aplique el algoritmo a cada uno de los grafos a, b y c. En cada caso muestre el estado de los grafos (valores de v.d para cada nodo, puede mostrarlos dentro de cada nodo omitiendo la etiqueta del nodo) después de cada iteración del for en la línea 2.

Respuesta:

Código:

```
class edge:

    def __init__(self, u, v, w):

        self.u = u
```

```
        self.v = v

        self.w = w


edgesa = [edge(0,1,9),

           edge(0,2,3),

           edge(2,1,2),

           edge(1,3,1),

           edge(2,3,6),

           edge(3,2,4),

           edge(2,4,6),

           edge(3,4,2)]


edgesb = [edge(0,1,9),

           edge(0,2,3),

           edge(2,1,2),

           edge(1,3,1),

           edge(2,3,6),

           edge(3,2,-4),

           edge(2,4,6),

           edge(3,4,2)]


edgesc = [edge(0,1,2),

           edge(0,2,8),

           edge(2,1,2),

           edge(1,3,3),

           edge(2,3,6),

           edge(3,2,4),

           edge(2,4,6),

           edge(3,4,2)]
```

```
def BellmanFord(vertices, edges, source, print_=False):
   distance = [0]*vertices
   predecessor = [0]*vertices

   for v in range(vertices):
       distance[v] = 10000000
       predecessor[v] = -1

   distance[source] = 0

   for i in range(vertices - 1):
       for edge in edges:
           u = edge.u
           v = edge.v
           w = edge.w
           if distance[u] + w < distance[v]:
               distance[v] = distance[u] + w
               predecessor[v] = u
       print(distance)

   for edge in edges:
       u = edge.u
       v = edge.v
       w = edge.w
       if (distance[u] + w < distance[v]):
           return "Graph contains a negative-weight cycle", 
distance, predecessor
   return distance, predecessor


print(BellmanFord(5, edgesa, 0))
print(BellmanFord(5, edgesb, 0))
```

```
print(BellmanFord(5, edgesc, 0))
```

Estados grafo a:

```
[0, 5, 3, 6, 8]

[0, 5, 3, 6, 8]

[0, 5, 3, 6, 8]

[0, 5, 3, 6, 8]

([0, 5, 3, 6, 8], [-1, 2, 0, 1, 3])
```

Estados grafo b:

```
[0, 5, 2, 6, 8]

[0, 4, 1, 5, 7]

[0, 3, 0, 4, 6]

[0, 2, -1, 3, 5]

('Graph contains a negative-weight cycle', [0, 2, -1, 3, 5], [-1,
2, 3, 1, 2])
```

Estados grafo c:

```
[0, 2, 8, 5, 7]

[0, 2, 8, 5, 7]

[0, 2, 8, 5, 7]

[0, 2, 8, 5, 7]

([0, 2, 8, 5, 7], [-1, 0, 0, 1, 3])
```

4. ¿Qué pasó con el grafo b?
   Respuesta: El grafo b contiene un ciclo negativo por lo tanto el algoritmo “Bellman-Ford” genera un error.

5. En cada caso, ¿cuántas llamadas se hicieron a la función Relax? ¿puede hacerse de manera más eficiente?
   Respuesta: En los tres grafos se le hicieron (vértices-1)*edges = 4*8 = 32 llamados a la sección de código que corresponde a la función relax. Debido a la construcción del algoritmo no se

pueden realizar menos llamados sin comprometer la correctitud del algoritmo. Sin embargo, se podría aplicar métodos voraces como el algoritmo de Dijkstra para realizar los llamados en un orden más eficiente y además cuando en alguna iteración no hayan ocurrido ningún cambio en las distancias significa que el algoritmo ha encontrado la respuesta.

6. Para los grafos a y c, muestre una secuencia de llamadas a Relax que le permita calcular los caminos más cortos de una manera más eficiente
Respuesta:

Para el grafo a:

0->2

2->1

1->3

3->4

el resto de arcos que se recorran no importa el orden ya que este ya habrá encontrado la respuesta óptima para cada nodo.

Para el grafo b:

0->1

1->3

3->2

2->1

3->4

Los dos últimos en cualquier orden