Problem Description

The problem is to find the shortest distance to all vertices from a source vertex in a weighted directed graph that can have negative edge weights. For the problem to be well-defined, there should be no cycles in the graph with a negative total weight.

Problem Solution

1. Create classes for Graph and Vertex.
2. Create a function bellman-ford that takes a Graph object and a source vertex as arguments.
3. A dictionary distance is created with keys as the vertices in the graph and their value all set to infinity.
4. distance[source] is set to 0.
5. The algorithm proceeds by performing an update operation on each edge in the graph n – 1 times. Here n is the number of vertices in the graph.
6. The update operation on an edge from vertex i to vertex j is distance[j] = min(distance[j], distance[i] + weight(i, j)).
7. The dictionary distance is returned.

Program/Source Code

Here is the source code of a Python program to implement Bellman-Ford algorithm on a graph. The program output is shown below.

```
# source https://www.sandry.com/

class Graph:
    def __init__(self):
        # dictionary containing keys that map to the corresponding vertex object
        self.vertices = {}

    def add_vertex(self, key):
        """Add a vertex with the given key to the graph."""
        vertex = Vertex(key)
        self.vertices[key] = vertex

    def get_vertex(self, key):
        """Return vertex object with the corresponding key."""
        return self.vertices[key]

    def __contains__(self, key):
        return key in self.vertices

    def add_edge(self, src_key, dest_key, weight=1):
        """Add edge from src_key to dest_key with given weight."""
        self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

    def does_edge_exist(self, src_key, dest_key):
        """Return True if there is an edge from src_key to dest_key."""
        return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

```

```
    def __len__(self):
        return len(self.vertices)

    def __iter__(self):
        return iter(self.vertices.values())


class Vertex:
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """Return key corresponding to this vertex object."""
        return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to dest."""
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to dest."""
        return dest in self.points_to


def bellman_ford(g, source):
    """Return distance where distance[v] is min distance from source to v.

    This will return a dictionary distance.

    g is a Graph object which can have negative edge weights.
    source is a Vertex object in g.
    """
    distance = dict.fromkeys(g, float('inf'))
    distance[source] = 0

    for _ in range(len(g) - 1):
        for v in g:
            for n in v.get_neighbours():
                distance[n] = min(distance[n], distance[v] + v.get_weight(n))

    return distance


g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> <weight>')
print('bellman-ford <source vertex key>')
print('display')
print('quit')

```

```
while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation == 'vertex':
            key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3])
            weight = int(do[4])
            if src not in g:
                print('Vertex {} does not exist.'.format(src))
            elif dest not in g:
                print('Vertex {} does not exist.'.format(dest))
            else:
                if not g.does_edge_exist(src, dest):
                    g.add_edge(src, dest, weight)
                else:
                    print('Edge already exists.')

    elif operation == 'bellman-ford':
        key = int(do[1])
        source = g.get_vertex(key)
        distance = bellman_ford(g, source)
        print('Distances from {}: '.format(key))
        for v in distance:
            print('Distance to {}: {}'.format(v.get_key(), distance[v]))
        print()

    elif operation == 'display':
        print('Vertices: ', end='')
        for v in g:
            print(v.get_key(), end=' ')
        print()

        print('Edges: ')
        for v in g:
            for dest in v.get_neighbours():
                w = v.get_weight(dest)
                print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                                             dest.get_key(), w))
        print()

    elif operation == 'quit':
        break
```

Program Explanation

1. An instance of Graph is created.
2. A menu is presented to the user to perform various operations on the graph.
3. To find all shortest distances from a source vertex, bellman-ford is called on the graph and the source vertex.

Runtime Test Cases

```
Case 1:
Menu
add vertex <key>
add edge <src> <dest> <weight>
bellman-ford <source vertex key>
display
quit
What would you like to do? add vertex 1
What would you like to do? add vertex 2
What would you like to do? add vertex 3
What would you like to do? add vertex 4
What would you like to do? add vertex 5
What would you like to do? add vertex 6
What would you like to do? add vertex 7
What would you like to do? add vertex 8
What would you like to do? add edge 1 2 10
What would you like to do? add edge 1 8 8
What would you like to do? add edge 2 6 2
What would you like to do? add edge 3 2 1
What would you like to do? add edge 3 4 1
What would you like to do? add edge 4 5 3
What would you like to do? add edge 5 6 -1
What would you like to do? add edge 6 3 -2
```

```
What would you like to do? add edge 7 2 -4
What would you like to do? add edge 7 6 -1
What would you like to do? add edge 8 7 1
What would you like to do? bellman-ford 1
Distances from 1:
Distance to 5: 9
Distance to 6: 7
Distance to 7: 9
Distance to 2: 5
Distance to 1: 0
Distance to 8: 8
Distance to 3: 5
Distance to 4: 6
```

```
Case 2:
Menu
add vertex <key>
add edge <src> <dest> <weight>
bellman-ford <source vertex key>
display
quit
What would you like to do? add vertex 1
What would you like to do? bellman-ford 1
Distances from 1:
Distance to 1: 0

What would you like to do? add vertex 2
What would you like to do? bellman-ford 1
Distances from 1:
Distance to 1: 0
Distance to 2: inf

What would you like to do? add edge 1 2 2
What would you like to do? add vertex 3
What would you like to do? add edge 1 3 -1
```

```
What would you like to do? bellman-ford 1
Distances from 1:
Distance to 1: 0
Distance to 3: -1
Distance to 2: 2

What would you like to do? add edge 3 2 2
What would you like to do? bellman-ford 1
Distances from 1:
Distance to 1: 0
Distance to 3: -1
Distance to 2: 1

What would you like to do? quit
```