**Practical work no. 3**

Bellman Ford’s algorithm:

**def** bellman(self, source, target):  
 previous = {}  
 distance = {}  
 queue = []  
 visited = {}  
 count = {}  
  
 cost\_dict = self.graph.get\_edges()  
  
 *#we initialize the dictionaries* **for** node **in** self.graph.get\_vertices():  
 distance[node] = math.inf  
 previous[node] = **None** count[node] = 0  
 *#no node is visited at the beginning* visited[node] = **False** *#the distance is 0 at the beginning* distance[source] = 0  
  
 *#push the source in the queue* queue.append(source)  
  
 *#the source vertex is marked as visited* visited[source] = **True  
  
 while** queue:  
 vertex = queue[0]  
 queue.pop(0)  
 visited[vertex] = **False** *#the vertex is marked as not visited* **for** child **in** self.get\_outbound(vertex): *#search through all its neoghbours* **if** distance[child] > distance[vertex] + cost\_dict[(vertex, child)]:

*#update the distance*  
 distance[child] = distance[vertex] + cost\_dict[(vertex, child)] previous[child] = vertex  
  
 **if** visited[child] == **False**:  
 visited[child] = **True** *#child is marked as true* count[child] += 1  
 queue.append(child) *#push the child in the queue  
  
 #if the number of vertices is greater than the maximum number of vertices, we have negative cost cycle* **if** count[child] >= len(self.graph.get\_out()):  
 print(**"Negative cost cycle!"**)  
 **return None** *#returns a pair distance, path* **return** (distance[target], self.get\_path(source, target, previous))

**def** get\_path(self, v1, v2, dict):  
 lista = []  
 node = v2  
 **while** node != v1:  
 lista.append(node)  
 node = dict[node]  
 lista.append(v1)  
  
 lista.reverse()  
 **return** lista