```
# I, Miguel Logarta,
# have neither given or received inappropriate help with this homework assignment.
from graph import Graph, Edge, Path
from verifyHalfSspD import verifyHalfSspD

DEV = False
VERBOSE = True
vhspd = 'Verbose: convertSspDToHalfSspD() --'

def make_new_node(nodes,new_nodes):
    '''
    Arguments:
      nodes -- list of nodes in the SspD instance
      new_nodes -- list of unique node names, none of which are in the SspD
instance.

    Returns a node name consisting of Z's, which is not in either nodes or
new_nodes
    '''

    new_node = 'Z'
    names = nodes + new_nodes
    for n in names:
        if n == new_node: new_node += 'Z'

    return new_node

def make_new_nodes(nodes):
    '''
    Arguments:
      nodes -- list of nodes in the SspD instance

    Returns a list of unique new node names that is long as the
    number of nodes in the Sspd instance.
    '''

    new_nodes = []
    for n in nodes:
        new_nodes.append(make_new_node(nodes,new_nodes))

    return new_nodes


def convertSspDToHalfSspD(SspD_instance): # I is an instance of SspD

    '''
    SspD_instance, an instance of the Stranded Salesperson decision problem, is a
    string with 3 parts:
      1.  G -- A weighted, undirected graph, specified as a white-space delimited
      list of edges. For example, 'b,a,2' is an edge of length 2
      between.

      2. A semicolon (';') separating G from L.

      3. L -- a non-negative integer representing the threshold, the maximum
      acceptable length of a Hamilton path

    SspD_instance is a positive instance of SspD if it has a Hamilton path
    whose length is no greater than L. For example, 'a,b,3 b,c,2 c,a,7;5' would
    be a positive instance of SspD because it has 3 nodes, a,b, & c, and path
    a-b-c has length 5, the same as the maximum acceptable path length.

    HalfSspD has the same syntax as SspD. An instance will be a positive
    instance of HalfSspD if it is a positive instance of SspD; or if it contains
    a subgraph with a Hamilton path whose length is no greater than the
    threshold, and the number of nodes in the subgraph is at least half
    of the number of nodes in the problem instance. For example,
    'a,b,3 b,c,2 d,e,1 f,e,1 ;6' is a positive instance of HalfSspD,
    because a-b-c connects half then nodes in the problem instance, and
    its length is 5, less than the maximum acceptable path length.

    This functions transforms SspD_instance into a HalfSspD instance named
    HalfSspD_instance, such that if SspD_instance is a positive instance of
    SspD, HalfSspD_instance will be a positive instance of HalfSspD; and if
    and if SspD_instance  is a negative instance of SspD, HalfSspD_instance
    will be a negative instance of HalfSspD.

    '''
    # split problem instance graph and threshold parts
    edges,L = SspD_instance.split(';')

    try:
        threshold = int(L)
    except:
        print(f'{vhspd}{L} is not a valid path threshold.')
        return 'unsure'

    # Use edges to create an undirected, weigthed WCBC graph object.
    #
    g = Graph(edges, directed = False)
    if DEV: print(f'DEV: Graph == {g}')

    nodes = g.nodes.keys()  # Get the node name keys from the graph object node
dictionary
    nodes = list(nodes)     #  and make it into a list

    # new_nodes will be a list of unique node_names, and there will be
    # as many items in this list as there were nodes in the SspD instance.
    # That is, len(new_nodes) == len(nodes)
    new_nodes = make_new_nodes(nodes)

    if DEV: print(f'DEV: New nodes == {new_nodes}')

    #### ***** L1010 -- Modify edges; taken from Sspd_instance, so that it can
    ###                 be used to create the appropriate HalfSspD_instance
    ####
    # edges = edges # L1010 change may require more than one line of code.
    # Note: I am not sure why the edge names are converted to Z, ZZ, ZZZ
    # It is failing test cases 1 and 4 because it is comparing the value
    # a, b, c to the value Z, ZZ, and ZZZ
    edges = edges.split(' ')
    half_sspd_edges = []
    for edge in edges:
        node_a, node_b, weight = edge.split(',')
        new_node_a = new_nodes[nodes.index(node_a)]
        new_node_b = new_nodes[nodes.index(node_b)]
        half_sspd_edges.append(f"{new_node_a},{new_node_b},{weight}")
    # HalfSspD_instance =  SspD_instance  # Change this for L1010
    HalfSspD_instance = f"{' '.join(half_sspd_edges)};{threshold}"
    print(HalfSspD_instance)
    if VERBOSE: print(f'{vhspd} HalfSspD_instance == {HalfSspD_instance}')

    return  HalfSspD_instance

C = convertSspDToHalfSspD # Using 'C' as the name of the conversion function

def vfySspDViaVfyHalfSspD(SspD_instance,S,H):
    # No need to modify this function for L1010

    # SspD_instance is a positive instance of SspD iff HalfSspD_instance
    # is a positive instance of HalfSspD.

    # Use the conversion function to create the appropriate HalfSSpD instance.
    HalfSspD_instance = C(SspD_instance)
    return verifyHalfSspD(HalfSspD_instance,S,H)

if __name__ == '__main__':

    def test_case(I,S,H,expected,num,comment=''):
        # No need to modify this function for L1010

        err = '** '
        result =  vfySspDViaVfyHalfSspD(I,S,H)
        func = f'''vfySspDViaVfyHalfSspD("{I}","{S}","{H}")'''
        if result == expected: err = ''
        e = expected
        print (f'{err}test #{num} {func}: expected "{e}", received "{result}"')
        print (f'test #{num} Explanation: {comment}\n')
        return num + 1

    num = 1
    exp  = 'Hint Hamiltonian path has length 5=threshold'
    num = test_case('a,b,3 b,c,2 c,a,7;5','Yes','a,b,c','correct',num,exp)

    exp  = 'Hint Hamiltonian path is longer than threshold'
    num = test_case('a,b,3 b,c,2 c,a,7;4','yes','a,b,c','unsure',num,exp)

    exp  = 'Hint is not a Hamiltonian path'
    num = test_case('a,b,3 b,c,2 c,d,7;15','yes','a,b,c','unsure',num,exp)

    exp  = 'Hint Hamiltonian path has length 14 < threshold'
    num = test_case('a,b,7 a,c,2 b,c,7;15','yes','a,b,c','correct',num,exp)
```