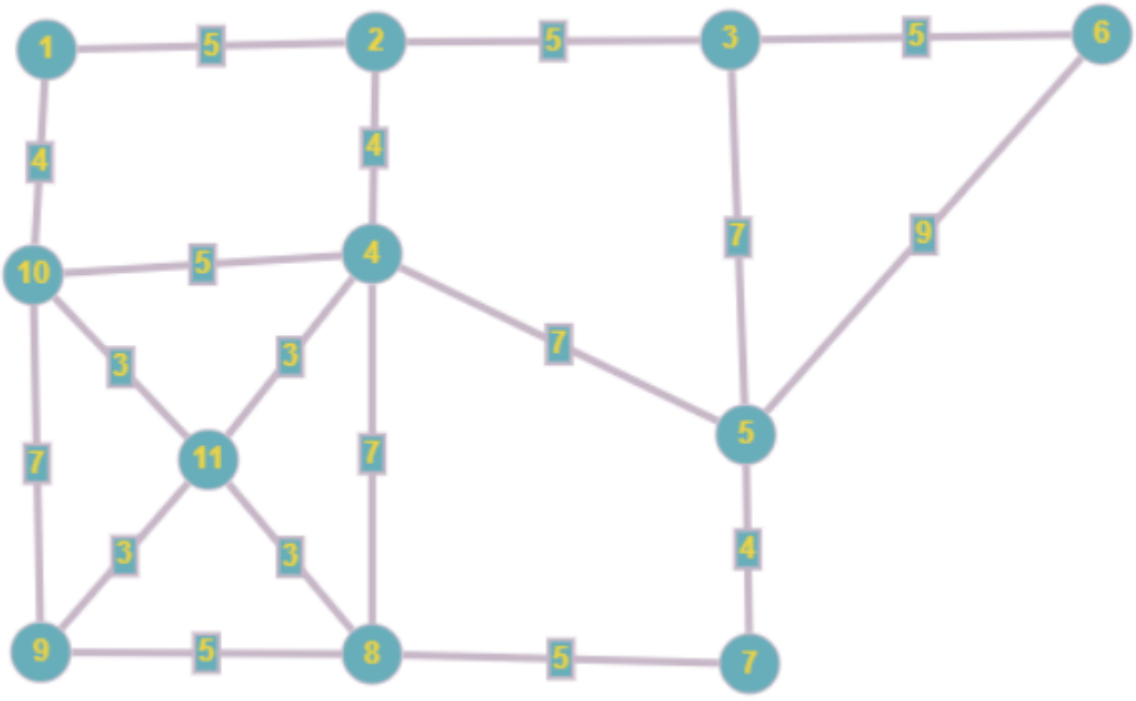
Objectif: Trouver un chemin Eulerien dans un graphe

1ère étape: Convertir le graphe en graphe Eulerien si cela n'est pas déjà le cas

Entrée [1]:

```
# graph initial simulant une partie de la ville
# (1er noeud, 2ème noeud, distance)
G = [(1,2,5), (1,10,4), (2,3,5), (2,4,4), (3,5,7), (3,6,5), (4,5,7), (4,8,7), (4,10,5), (4,
     (7,8,5), (8,9,5), (8,11,3), (9,10,7), (9,11,3), (10,11,3)]
print("Graphe initial:\n", G)
```

```
Graphe initial:
 [(1, 2, 5), (1, 10, 4), (2, 3, 5), (2, 4, 4), (3, 5, 7), (3, 6, 5), (4, 5,
7), (4, 8, 7), (4, 10, 5), (4, 11, 3), (5, 6, 9), (5, 7, 4), (7, 8, 5), (8,
9, 5), (8, 11, 3), (9, 10, 7), (9, 11, 3), (10, 11, 3)]
```



Entrée [2]:

```
# return the list of odd vertices
def odd_vertices(n, edges):
    deg = [0] * n
    for (a,b,c) in edges:
        deg[a] += 1
        deg[b] += 1
    return [a for a in range(n) if deg[a] % 2]

# convert the graph to an adjacency list
def graph_to_adj(n, G):
    adj = [[] for _ in range(n)]
    for (a, b, w) in G:
        adj[a].append((b,w))
        adj[b].append((a,w))
    return adj

# application of dijkstra's algorithm to find the shortest path from a certain vertex to al
def dijkstra(G, s, path, n):
    adj = graph_to_adj(n, G)
    infi = 100000000
    dist = [infi for _ in range(len(adj))]
    visited = [False for _ in range(len(adj))]
    for i in range(len(adj)):
        path[i] = -1
    dist[s] = 0
    path[s] = -1
    current = s
    sett = set()
    while (True):
        visited[current] = True
        for i in range(len(adj[current])):
            v,_ = adj[current][i];
            if (visited[v]):
                continue
            sett.add(v)
            alt = dist[current] + adj[current][i][1]
            if (alt < dist[v]):
                dist[v] = alt
                path[v] = current;
        if current in sett:
            sett.remove(current);
        if (len(sett) == 0):
            break
        minDist = infi
        index = 0
        for a in sett:
            if (dist[a] < minDist):
                minDist = dist[a]
                index = a;
        current = index
    return dist

# find the closest odd vertex to a certain vertex
def find_closest_odd(v, odd, dist):
    closest = odd[1]
    min_val = dist[closest]
    for v in odd[2:]:
        if dist[v] < min_val:
            closest = v
```

```
            min_val = dist[v]
    return closest, min_val

# convert the graph to an eulerian graph
def to_eulerian(n, edges):
    odd = odd_vertices(n, edges)
    while len(odd) != 0 and len(odd) != 2:
        v = odd[0]
        path = [0 for _ in range(len(edges))]
        dist = dijkstra(edges, v, path, n)
        closest, w = find_closest_odd(v, odd, dist)
        edges.append((v, closest, w))
        odd.remove(v)
        odd.remove(closest)
    return edges

# extract all vertices from a list of edges
def extract_vertices(edges):
    vertices = []
    for (a,b,_) in edges:
        if a not in vertices:
            vertices.append(a)
        if b not in vertices:
            vertices.append(b)
    return vertices
```

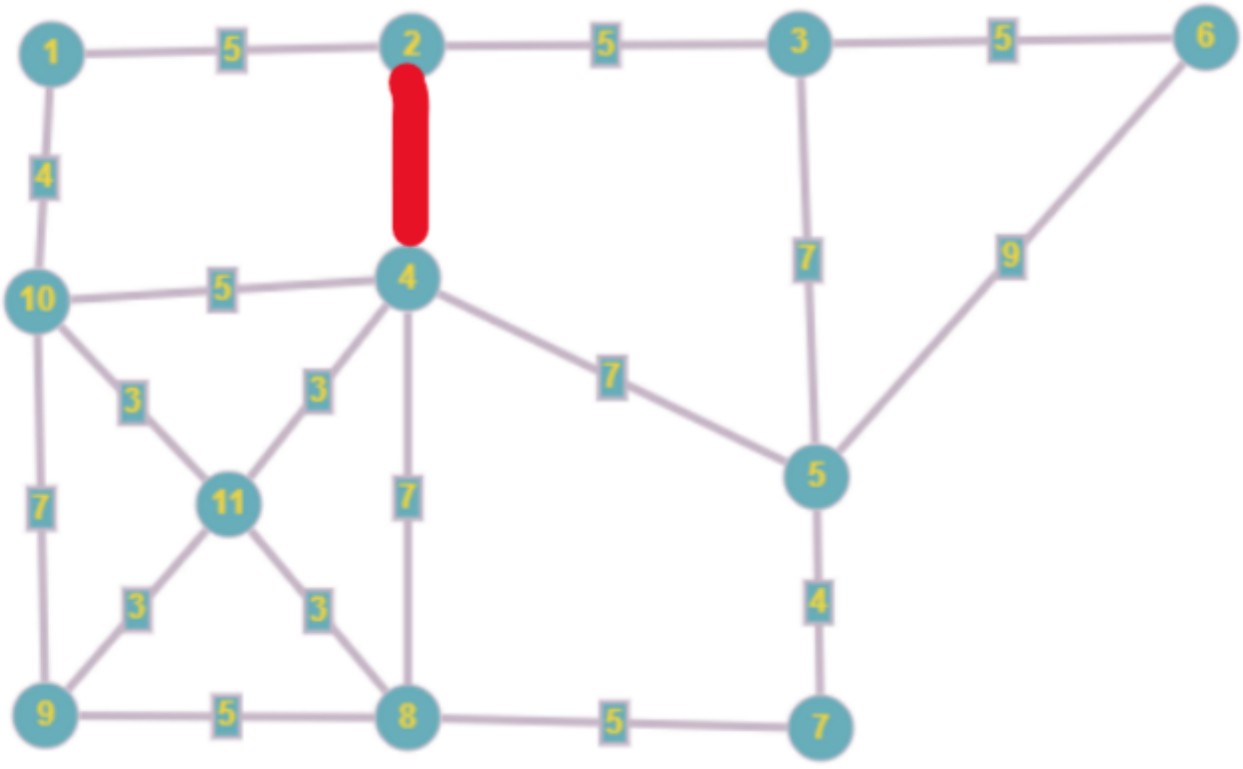
Entrée [3]:

```
l = extract_vertices(G)
G = to_eulerian(len(l)+1, G)
print("Graphe Eulerien:\n", G)
print("Nombre de noeuds impaires:", len(odd_vertices(len(l)+1, G)))
```

```
Graphe Eulerien:
 [(1, 2, 5), (1, 10, 4), (2, 3, 5), (2, 4, 4), (3, 5, 7), (3, 6, 5), (4, 5,
7), (4, 8, 7), (4, 10, 5), (4, 11, 3), (5, 6, 9), (5, 7, 4), (7, 8, 5), (8,
9, 5), (8, 11, 3), (9, 10, 7), (9, 11, 3), (10, 11, 3), (2, 4, 4)]
Nombre de noeuds impaires: 2
```

On peut voir qu'une nouvelle arête à été ajouté au graphe.

Celle-ci représente le chemin le plus court entre deux noeuds impaires pour que le nombre de noeuds impaires du graphe soit <= 2 et que celui-ci soit donc eulerien.



2ème étape: Trouver un chemin Eulerien dans le graphe

Entrée [4]:

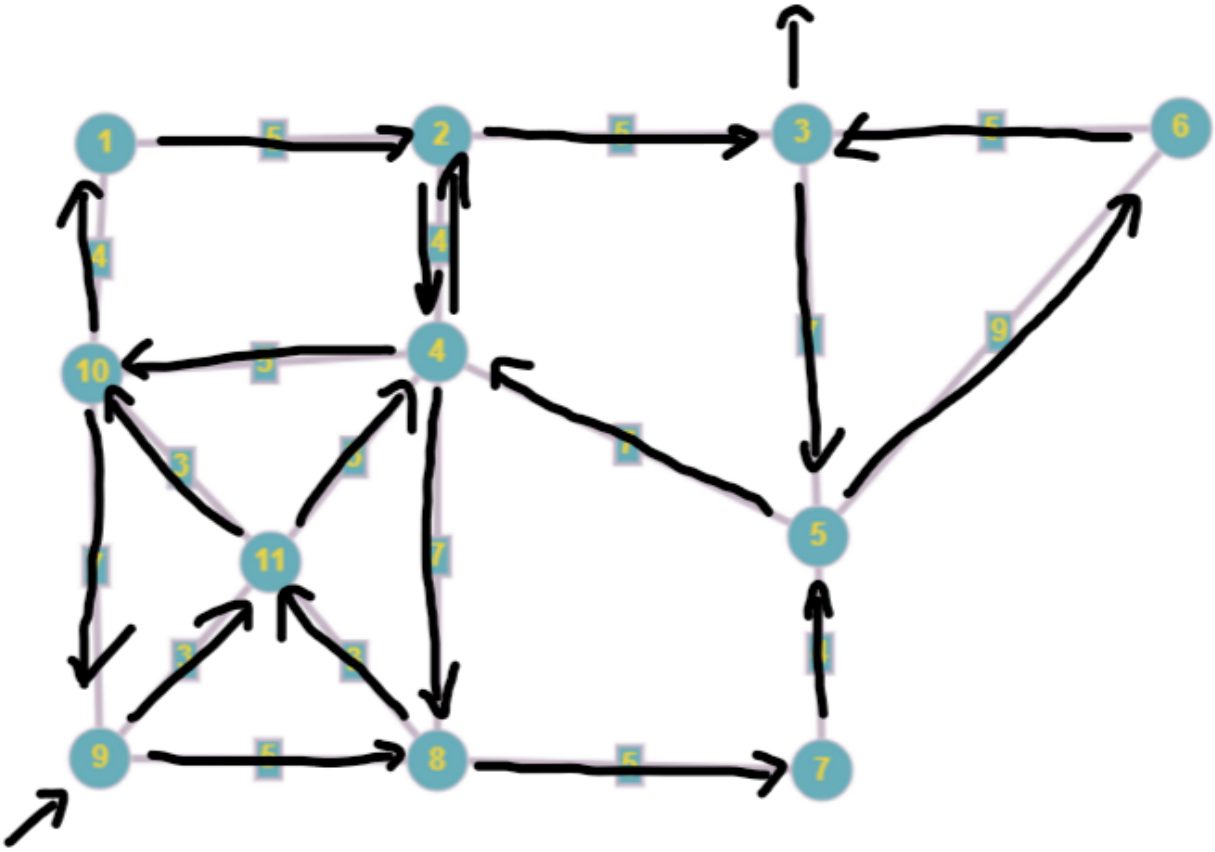
```
# find the edge with a certain vertex
def find_edge(edges, v):
    for (a,b,c) in edges:
        if a == v or b == v:
            return (a,b,c)
    return None

# find the eulerian path
def find_eulerian_path(n, edges):
    odd = odd_vertices(n, edges)
    if len(odd) == 2:
        stack = [odd[0]]
    else:
        stack = [edges[0][0]]
    path = []
    while stack:
        v = stack[-1]
        edge = find_edge(edges, v)
        if edge:
            u = edge[0]
            if u == v:
                u = edge[1]
            stack.append(u)
            edges.remove(edge)
        else:
            path.append(stack.pop())
    return path
```

Entrée [5]:

```
print("Chemin Eulerien du graphe:\n", find_eulerian_path(len(G), G))
```

```
Chemin Eulerien du graphe:
 [9, 11, 10, 9, 8, 11, 4, 8, 7, 5, 6, 3, 5, 4, 2, 4, 10, 1, 2, 3]
```



On peut voir avec le dessin ci-dessus que nous avons bien un chemin eulerien (si on compte la nouvelle arête créée comme une arête bien distincte) et donc que notre algorithme marche correctement pour un graphe simple.