COP 5537 ASSIGNMENT #2

1 Pseudocode for Fleury's Algorithm

Algorithm 1 Fleury's_Algorithm (Adjacency matrix A, Vertex start, node_count)

```
V ← {v1, v2, ....., vn }                                   ▷ Set of vertices
vertex_count ← node_count                                   ▷ Assign current number of vertices in graph A

for v ← 0 to n do
    if A[start][v] then                                     ▷ Checks if edge (start, v) exists
        visited_nodes ← 0                                   ▷ Initialize all vertices as non-visited

        A[start][v] = A[v][start] = 0                       ▷ Temporarily removing edge (start, v) to check if graph gets disconnected
        visited_count = Prims(A, n, v, visited_nodes)       ▷ Counting number of connected vertices using Prim's algorithm
        if abs(vertex_count - visited_count) ≤ 1 then       ▷ Removing edge (start, v) will not separate A into two
                                                              disconnected sets of edges
            print edge (start, v)

            if Bridge_edge(start, v) then                   ▷ Edge (start,v) is bridge edge
                vertex_count = vertex_count - 1             ▷ Decrease vertex count by 1
            if Bridge_edge(v, start) then                   ▷ Edge (v,start) is bridge edge
                vertex_count = vertex_count - 1             ▷ Decrease vertex count by 1

            A[start][v] = A[v][start] = 0                   ▷ Removing edge (start, v)
            Fleury's_Algorithm (A, v, vertex_count)
        else
            A[start][v] = A[v][start] = 1                   ▷ Not removing edge (start, v)
```

Algorithm 2 Prims(Adjacency matrix A, n, Vertex source, visited_nodes)

```
distance ← ∞                                                ▷ Initialization
parent ← -1
count ← 0
distance[source] ← 0                                        ▷ Set distance of source as 0

for k ← 1 to n do
    u ← Min_dis(distance, visited_nodes)                    ▷ Returns node u with the shortest distance from source; if no
                                                              connected node from source then returns -1
    if u ≠ -1 then                                          ▷ Vertex u is connceted and visited
        visited_nodes[u] = 1
        count = count + 1

        for v ← 1 to n do
            if visited_nodes[v] = 0 and A[u][v] > 0 and A[u][v] < distance[v] then
                distance[v] = A[u][v]
                parent[v] = u
return count
```