Pseudocode

Input: A graph G and a vertex v of G.

Output: A labeling of the edges in the connected component of v as discovery edges and back edges.

```
def DFS(G, v, explored, discovery_edges, back_edges):
    # Mark vertex v as explored
    explored[v] = True

    # Loop through all incident edges of vertex v
    for e in G.incidentEdges(v):
        if not explored[e]:
            # Get the adjacent vertex connected by edge e
            w = G.adjacentVertex(v, e)

            if not explored[w]:
                # Label the edge as a discovery edge
                discovery_edges.append(e)
                # Recursively call DFS on the adjacent vertex w
                DFS(G, w, explored, discovery_edges, back_edges)
            else:
                # Label the edge as a back edge
                back_edges.append(e)

# Example usage
class Graph:
    def __init__(self, num_vertices):
        self.num_vertices = num_vertices
        # Initialize data structures for the graph representation

    def incidentEdges(self, vertex):
        # Return a list of incident edges for the given vertex
```

```
    def adjacentVertex(self, vertex, edge):
        # Return the adjacent vertex for the given vertex and edge

# Create a graph
num_vertices = 7  # Example number of vertices
G = Graph(num_vertices)

# Initialize data structures
explored = [False] * num_vertices
discovery_edges = []
back_edges = []

# Choose a starting vertex
start_vertex = 0

# Call DFS to explore the graph and label edges
DFS(G, start_vertex, explored, discovery_edges, back_edges)

# Print the labeled edges
print("Discovery Edges:", discovery_edges)
print("Back Edges:", back_edges)
```