Breadth First Search

```
from collections import deque

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)

    def bfs(self, start_vertex):
        visited = {vertex: False for vertex in self.graph}
        queue = deque()
        queue.append(start_vertex)
        visited[start_vertex] = True

        while queue:
            vertex = queue.popleft()
            print(vertex, end=' ')

            for neighbor in self.graph[vertex]:
                if not visited[neighbor]:
                    queue.append(neighbor)
                    visited[neighbor] = True


# Example usage
g = Graph()
g.add_edge(1, 2)
g.add_edge(1, 3)
g.add_edge(2, 4)
g.add_edge(2, 5)
g.add_edge(3, 6)
g.add_edge(3, 7)

print("BFS starting from vertex 1:")
g.bfs(1)
```

```
BFS starting from vertex 1:
1 2 3 4 5 6 7
```

Depth First Search

```
class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)

    def dfs_recursive(self, vertex, visited):
        visited[vertex] = True
        print(vertex, end=' ')

        for neighbor in self.graph[vertex]:
            if not visited[neighbor]:
                self.dfs_recursive(neighbor, visited)

    def dfs(self, start_vertex):
        visited = {vertex: False for vertex in self.graph}
        self.dfs_recursive(start_vertex, visited)


g = Graph()
g.add_edge(1, 2)
g.add_edge(1, 3)
g.add_edge(2, 4)
g.add_edge(2, 5)
g.add_edge(3, 6)
g.add_edge(3, 7)

print("DFS starting from vertex 1:")
g.dfs(1)
```

```
DFS starting from vertex 1:
1 2 4 5 3 6 7
```