```python
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt


def plot_frame(G, colormap, ax):
    """Plots on a figure a frame of colored graph

    Parameters:
    G (nx Graph)        : the graph to be plotted
    colormap (dict)     : a dictionary containing node-color pairs
    ax (plt axes)       : the figure on which to draw

    Returns:
    None
   """
    ax.clear()
    pos_layout = nx.spring_layout(G, seed=1)
    colors = [colormap[v] for v in G]
    nx.draw_networkx(G, with_labels=True, pos=pos_layout, node_color=colors)
    plt.pause(1)


def visit(G, vs, animation, draw_graph_and_tree, alfa):
    """Visit a graph

    Parameters:
    G (nx Graph)        : the graph to be visited
    vs (nx Node)        : the starting node
    animation (boolean) : user decides whether to display the animation
    draw_graph_and_tree (boolean) : user decides whether to display the graph and
the spanning tree
    alfa (int)          : parmater used in pop and append operation on the list.
                            For alfa=0 we get breadth_first_visit
                            For alfa=-1 we get depth_first_visit

    Returns:
    T (nx Graph)    : spanning tree of the visit
   """
    if animation:
        plt.figure()
        ax = plt.axes()
        colormap = {v: 'blue' for v in G}
        plot_frame(G, colormap, ax)

    Q = []
    T = nx.Graph()
    Q.append(vs)
    G.nodes[vs]['found'] = True
    T.add_node(vs)
    while Q:
        vgreen = Q.pop(alfa)
```

```
        if animation:
            colormap[vgreen] = 'green'
        for v in G[vgreen]:
            if animation and colormap[v] == 'blue':
                colormap[v] = 'yellow'
            if 'found' not in G.nodes[v]:
                Q.append(v)
                G.nodes[v]['found'] = True
                T.add_node(v)
                T.add_edge(vgreen, v)
        if animation:
            if Q:
                colormap[Q[alfa]] = 'orange'
            plot_frame(G, colormap, ax)
            colormap[vgreen] = 'red'

    if animation:
        plot_frame(G, colormap, ax)
        plt.close()

    if draw_graph_and_tree:
        pos_layoutG = nx.spring_layout(G, seed=1)
        pos_layoutT = nx.spring_layout(T, seed=1)
        plt.figure()
        nx.draw_networkx(G, with_labels=True, pos=pos_layoutG)
        plt.figure()
        nx.draw_networkx(T, with_labels=True, pos=pos_layoutT)
        plt.show()

    for v in G:
        del G.nodes[v]['found']

    return T


def breadth_first_visit(G, vs, animation=False, draw_graph_and_tree=False):
    visit(G, vs, animation, draw_graph_and_tree, 0)


def depth_first_visit(G, vs, animation=False, draw_graph_and_tree=False):
    visit(G, vs, animation, draw_graph_and_tree, -1)


G = nx.cycle_graph(10)
G.add_nodes_from([10, 11])
G.add_edges_from([(10, 0), (10, 4), (10, 11)])
G.add_edges_from([(11, 5), (11, 7)])
G.add_edges_from([(1, 9), (2, 4), (6, 8)])

breadth_first_visit(G, 0, animation=True, draw_graph_and_tree=True)
depth_first_visit(G, 0, animation=True, draw_graph_and_tree=True)

```