PRACTICAL 1.1

Aim:- Implement the Breadth First Search algorithm to solve a given problem.

CODE:

```
import folium
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut

# Define the graph with node coordinates (latitude and longitude) for Maharashtra cities
dict_gn = {
   'Mumbai': {'Pune': 150, 'Nashik': 200, 'Aurangabad': 330, 'Nagpur': 800, 'Goa': 460},
   'Pune': {'Mumbai': 150, 'Nashik': 200, 'Aurangabad': 230, 'Nagpur': 650},
   'Nashik': {'Mumbai': 200, 'Pune': 200, 'Aurangabad': 220, 'Nagpur': 600},
   'Aurangabad': {'Mumbai': 330, 'Pune': 230, 'Nashik': 220, 'Nagpur': 500},
   'Nagpur': {'Mumbai': 800, 'Pune': 650, 'Nashik': 600, 'Aurangabad': 500},
   'Goa': {'Mumbai': 460},
   'Kolhapur': {'Pune': 230, 'Goa': 200},
   'Sangli': {'Kolhapur': 90, 'Pune': 260},
   'Satara': {'Pune': 110, 'Kolhapur': 100},
   'Solapur': {'Pune': 250, 'Aurangabad': 180},
   'Jalgaon': {'Nashik': 90, 'Aurangabad': 130},
   'Akola': {'Nagpur': 150, 'Aurangabad': 250},
   'Amravati': {'Nagpur': 150, 'Akola': 80},
   'Latur': {'Aurangabad': 200, 'Solapur': 120},
   'Parbhani': {'Aurangabad': 110, 'Latur': 100},
   'Osmanabad': {'Solapur': 90, 'Latur': 120},
   'Bhandara': {'Nagpur': 80},
   'Wardha': {'Nagpur': 70, 'Amravati': 110},
   'Chandrapur': {'Nagpur': 150},
   'Yavatmal': {'Amravati': 100, 'Nagpur': 160},
   'Malkapur': {'Jalgaon': 120, 'Buldhana': 80},
   'Buldhana': {'Malkapur': 80, 'Akola': 60},
   'Washim': {'Akola': 100, 'Buldhana': 50},
   'Jalna': {'Aurangabad': 50, 'Parbhani': 100},
   'Nanded': {'Parbhani': 80, 'Latur': 120},
   'Beed': {'Aurangabad': 120, 'Solapur': 180},
   'Raund': {'Pune': 250, 'Kolhapur': 130},
   'Dhule': {'Nashik': 120, 'Jalgaon': 130},
}
# Function to get coordinates using geopy
def get_coordinates(city_name):
   geolocator = Nominatim(user_agent="city_coordinates")
   try:
      location = geolocator.geocode(city_name + ", Maharashtra, India")
      if location:
         return (location.latitude, location.longitude)
      else:
         return (None, None)
   except GeocoderTimedOut:
      return get_coordinates(city_name)  # Retry if timed out
```

```
# Create a graph object
G = nx.Graph()

# Add edges to the graph
for node, neighbors in dict_gn.items():
    for neighbor, weight in neighbors.items():
        G.add_edge(node, neighbor, weight=weight)

# Fetch coordinates for all cities
coordinates = {}
for city in dict_gn.keys():
    lat, lon = get_coordinates(city)
    if lat is not None and lon is not None:
        coordinates[city] = (lat, lon)
    else:
        print(f"Warning: Coordinates for {city} not found.")
# Perform BFS to determine levels or distances
def bfs_levels(start):
    levels = {start: 0}
    queue = [start]
    while queue:
        node = queue.pop(0)
        current_level = levels[node]
        for neighbor in G.neighbors(node):
            if neighbor not in levels:
                levels[neighbor] = current_level + 1
                queue.append(neighbor)
    return levels

# Calculate node levels
levels = bfs_levels('Mumbai')

# Create a color map based on node levels
unique_levels = list(set(levels.values()))
color_map = plt.get_cmap('viridis')  # A colormap with a gradient
norm = plt.Normalize(min(unique_levels), max(unique_levels))
color_dict = {level: color_map(norm(level)) for level in unique_levels}

# Map node levels to colors
node_colors = [color_dict[levels[node]] for node in G.nodes()]

# Create the base map using folium
m = folium.Map(location=[19.0760, 72.8777], zoom_start=7)  # Center map roughly in Maharashtra

# Add nodes to the map
for city, (lat, lon) in coordinates.items():
    folium.CircleMarker(
        location=[lat, lon],
        radius=8,
        color=mcolors.to_hex(node_colors[list(coordinates.keys()).index(city)]),
        fill=True,
        fill_color=mcolors.to_hex(node_colors[list(coordinates.keys()).index(city)]),
        fill_opacity=0.7,
        tooltip=city
    ).add_to(m)
```

```
# Add edges to the map, only if both nodes have coordinates
for u, v in G.edges():
    if u in coordinates and v in coordinates:
        folium.PolyLine(
            locations=[coordinates[u], coordinates[v]],
            color='gray',
            weight=2,
            opacity=0.5
        ).add_to(m)

# Save the map to an HTML file
m.save('maharashtra_graph_map.html')
```

OUTPUT:-