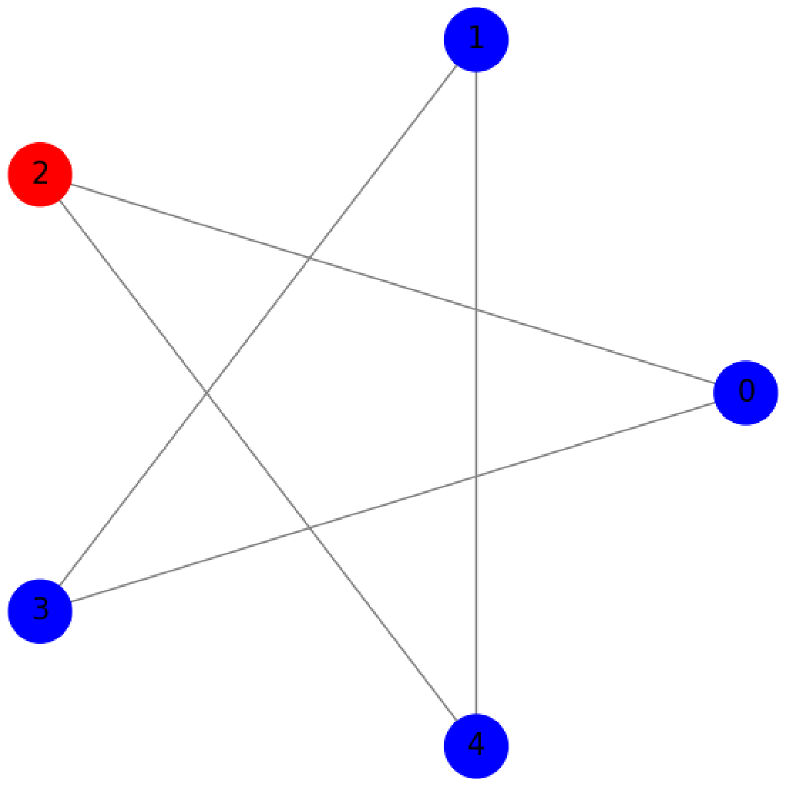
Spreading information on star grid



```
import random
import time
import matplotlib.pyplot as plt
import networkx as nx
import math
import numpy as np

def generate_star_graph_matrix(G):
    a = [[0 for _ in range(G)] for _ in range(G)]
    for i in range(G):
        a[i][(i + 2) % G] = 1.0
        a[(i + 2) % G][i] = 1.0
    return a

def visualize_graph(matrix, start_node):
    G = len(matrix)
    graph = nx.Graph()
    for i in range(G):
        for j in range(G):
            if matrix[i][j] == 1.0:
                graph.add_edge(i, j)

    pos = {i: (math.cos(2 * i * 3.14159 / G), math.sin(2 * i * 3.14159
/ G)) for i in range(G)}

    plt.figure(figsize=(6, 6))
    nx.draw(graph, pos, with_labels=True, node_color='lightblue',
node_size=1000, font_size=15, edge_color='gray')
    plt.title(f"Five-pointed Star Graph Visualization (Start Node:
{start_node})")
    plt.show()

def simulate_diffusion(G, matrix, p, iterations):
    M = [0.0] * G
    start_node = random.randint(0, G - 1)
    M[start_node] = 1.0

    pave_values = []
    fixation_time = -1

    for t in range(iterations):
        pave = sum(M) / G
        pave_values.append(pave)

        if pave == 1.0 and fixation_time == -1:
            fixation_time = t

        pi = random.randint(0, G - 1)
        while True:
            pf = random.randint(0, G - 1)
            if matrix[pi][pf] != 0:
                break

        if random.random() < p:
            M[pf] = M[pi]

    return pave_values, fixation_time, pave_values[-1] == 1.0

def calculate_fixation_metrics(G, matrix, p_values, iterations,
num_runs):
    fixation_probabilities = []
    fixation_times = []

    for p in p_values:
        fixation_count = 0
        total_fixation_time = 0

        fixation_time_values = []

        for _ in range(num_runs):
            _, fixation_time, fixation_occurred =
simulate_diffusion(G, matrix, p, iterations)
```

```
            if fixation_occurred:
                fixation_count += 1
                total_fixation_time += fixation_time
                fixation_time_values.append(fixation_time)

        fixation_probability = fixation_count / num_runs
        average_fixation_time = np.mean(fixation_time_values) if
fixation_time_values else 0

        fixation_probabilities.append(fixation_probability)
        fixation_times.append(average_fixation_time)

    return fixation_probabilities, fixation_times

def plot_fixation_metrics(p_values, fixation_probabilities,
fixation_times):
    plt.figure(figsize=(10, 6))
    plt.plot(p_values, fixation_probabilities, marker='o',
label="Fixation Probability")
    plt.title("Fixation Probability vs. Probability (p)")
    plt.xlabel("Probability (p)")
    plt.ylabel("Fixation Probability")
    plt.ylim(0, 1)
    plt.grid(True)
    plt.legend()
    plt.show()

    plt.figure(figsize=(10, 6))
    plt.plot(p_values, fixation_times, marker='o', label="Average
Fixation Time")
    plt.title("Average Fixation Time vs. Probability (p)")
    plt.xlabel("Probability (p)")
    plt.ylabel("Average Fixation Time")
    plt.grid(True)
    plt.legend()
    plt.show()

def main():
    random.seed(None)

    G = 5
    iterations = 500
    num_runs = 5000

    matrix = generate_star_graph_matrix(G)

    start_node = random.randint(0, G - 1)
    visualize_graph(matrix, start_node)

    p_values = [0.1 * i for i in range(1, 11)]
    fixation_probabilities, fixation_times =
calculate_fixation_metrics(G, matrix, p_values, iterations, num_runs)
```

```
    plot_fixation_metrics(p_values, fixation_probabilities,
fixation_times)

if __name__ == "__main__":
    main()
```