Graph Coloring Problem

Explain the Luby's Algorithm (Randomized Distributed Coloring algorithms for the problem:

Graph coloring is a classical problem in computer science and combinatorics, where the goal is to color the vertices of a graph such that no two adjacent vertices share the same color. Luby's Algorithm is a randomized approach to graph coloring that operates in parallel. It works by iteratively selecting a maximal independent set (MIS) of vertices and assigning them colors. This project explores Luby's Algorithm, a parallel and randomized method to solve the graph coloring problem efficiently. To investigate its implementation, runtime performance, and complex behavior.

Sketch of the Algorithm:

1. **Initialize:** Assign each vertex a unique random number.
2. **Independent Set Selection:** In parallel, each vertex checks if its number is higher than all its neighbors. If true, it joins the independent set.
3. **Color Assignment:** Assign a new color to all vertices in the independent set.
4. **Graph Reduction:** Remove colored vertices and their neighbors from the graph.
5. **Repeat:** Apply steps 2-4 to the remaining graph until all vertices are colored.

Data Structures Used:

The algorithm used this structure to help achieve fast access and mutation operations required in the parallel steps of the algorithm.

- **Graph Representation**: We use NetworkX's adjacency list for efficient neighborhood lookups.
- **Sets**: To maintain uncolored nodes and track the independent set in each round.
- **Dictionaries**: To store the randomly assigned priorities and color assignments.

Experimental Setup:

Random graphs were generated using the Erdős-Rényi model with vertex counts from 100 to 1000 (step size 100) and edge probability set to 0.05. Each graph was processed using the implemented Luby's Algorithm in Python. Runtime was measured using time.time(). Now to write the source code for better understanding.

Source code for this algorithm:

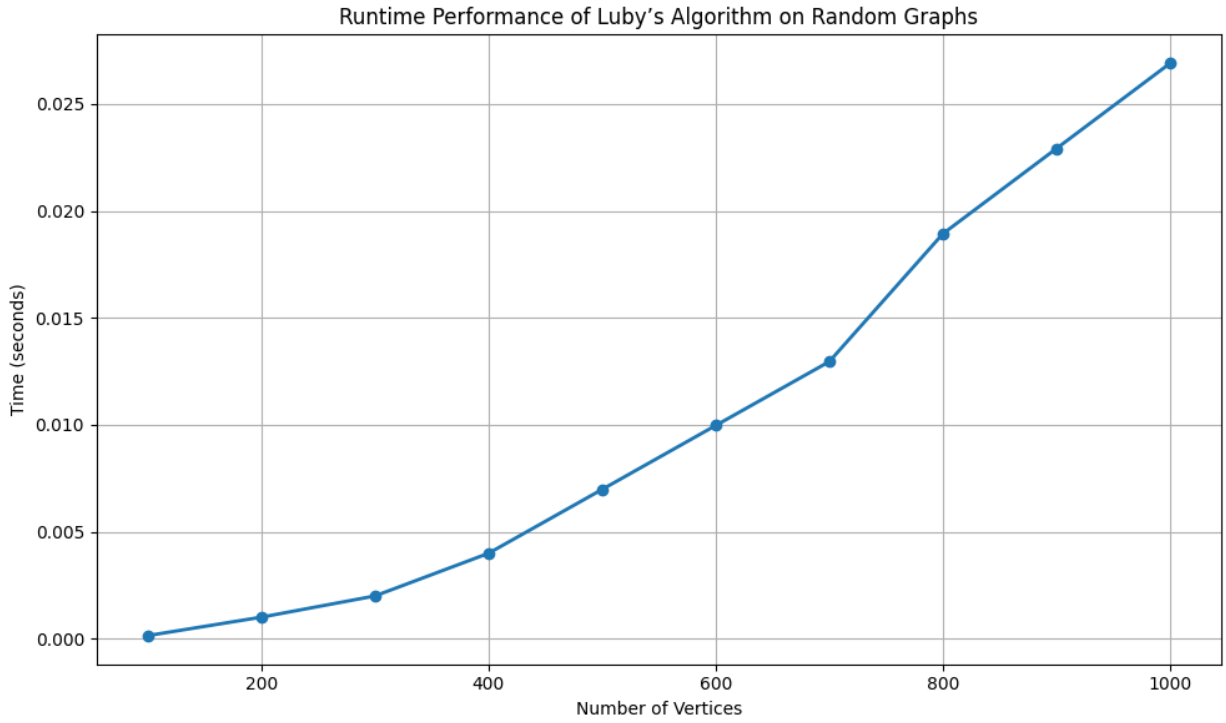
```
import networkx as nx
import random
import time
import matplotlib.pyplot as plt
def luby_graph_coloring(graph):
    color_assignment = {}
    uncolored_nodes = set(graph.nodes())
    current_color = 0

    while uncolored_nodes:
        # Assign random priority values
        priorities = {node: random.random() for node in uncolored_nodes}
        independent_set = set()

        # Select nodes with highest priority among their uncolored neighbors
        for node in uncolored_nodes:
            if all(priorities[node] > priorities.get(neigh, -1) for neigh in
graph.neighbors(node) if neigh in uncolored_nodes):
                independent_set.add(node)
        # Assign the current color to all nodes in the independent set
        for node in independent_set:
            color_assignment[node] = current_color
        # Remove colored nodes from the uncolored set
        uncolored_nodes -= independent_set
        current_color += 1
    return color_assignment
# Experimental run
input_sizes = list(range(100, 1100, 100))
runtimes = []
for size in input_sizes:
    G = nx.erdos_renyi_graph(n=size, p=0.05)  # Sparse random graph
    start = time.time()
    _ = luby_graph_coloring(G)
    end = time.time()
    runtimes.append(end - start)
print("Runtime", runtimes)
```

Result Analysis:

The runtime data is shown in the graph below:

Runtime [0.00013875961303710938, 0.0010013580322265625, 0.0019931793212890625, 0.003988504409790039, 0.00697636604309082, 0.009967327117919922, 0.01295614242553711, 0.0189361572265625, 0.02292346954345703, 0.026909589767456055]

The plot indicates a nearly linear or sub-quadratic growth pattern in runtime with respect to graph size, which aligns with the algorithm's expected behavior for sparse graphs.



Complexity Analysis:

- **Time Complexity**: O(log n) iterations, with each iteration requiring O(m) work in parallel (where m is the number of edges).
- **Space Complexity**: O(n + m) to store the graph and auxiliary data structures.

For dense graphs, runtime can increase more sharply, but for most sparse practical graphs, performance remains efficient. Luby's Algorithm offers a compelling solution for parallel graph coloring problems. Our implementation confirms its efficiency for random sparse graphs, showing manageable runtime growth with input size. This approach and its randomized behavior make it particularly suitable for distributed systems and large-scale graphs.