Graph Coloring Problem

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

Graph coloring is a fundamental problem in computer science and discrete mathematics, where the objective is to assign colors to the vertices of a graph such that no two adjacent vertices share the same color. This problem has numerous real-world applications including register allocation in compilers, frequency assignment in wireless networks, and task scheduling. In this project, to implement and evaluate Luby's Algorithm, a well-known parallel and randomized method for graph coloring. The algorithm operates by iteratively selecting a maximal independent set of vertices and assigning a color to each selected set. Its parallel nature and randomized selection make it suitable for large-scale and distributed graph processing.

Now, to focus on implementing Luby's Algorithm in an efficient manner using Python and NetworkX. The implementation is evaluated on randomly generated graphs of varying sizes to analyze the runtime performance. This report details the algorithm's structure, the data structures used, the empirical results, and the theoretical analysis of its time and space complexity.

Luby's Algorithm: This is a parallel randomized algorithm that solves the graph coloring problem by iteratively finding and coloring a maximal independent set (MIS). The steps are as follows:

1. Assign each uncolored vertex a random priority value.
2. In parallel, select all vertices that have the highest priority among their uncolored neighbors to form a maximal independent set.
3. Assign a new color to all vertices in the MIS.
4. Remove the newly colored vertices from the set of uncolored nodes.
5. Repeat steps 1–4 until all vertices are colored.

This strategy ensures that no two adjacent vertices are ever selected in the same round, thus preserving proper coloring.

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.

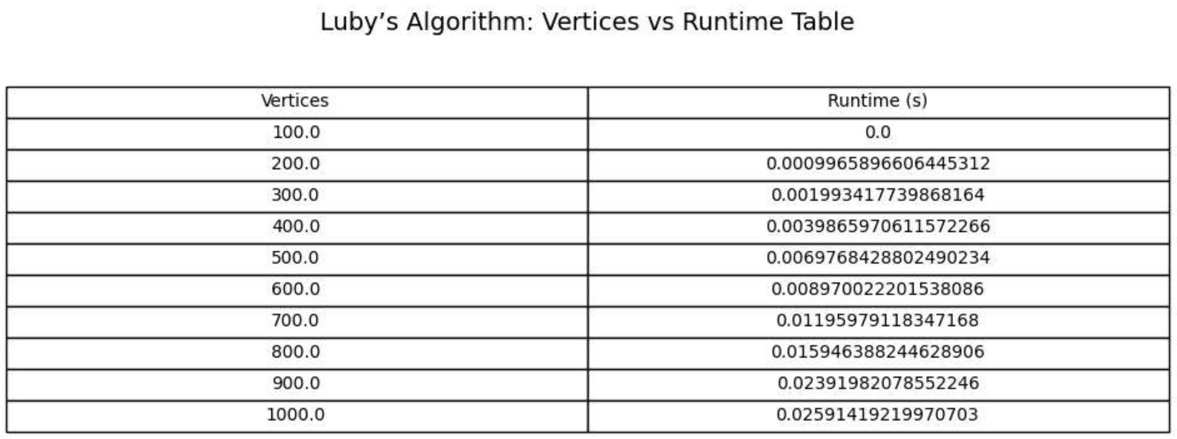
Experimental Setup: Random graphs were generated using the Erdős-Rényi model with vertex counts from 100 to 1000 (step size 100) and edge probability set to 0.05. Each graph was processed using the implemented Luby's Algorithm in Python. Runtime was measured using time.time(). Now to write the source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

Source code for this algorithm:

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

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

        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)

        for node in independent_set:
            color_assignment[node] = current_color

        uncolored_nodes -= independent_set
        current_color += 1

    return color_assignment
# Benchmark and capture outputs
input_sizes = list(range(100, 1100, 100))
runtimes = []
vertex_counts = []

results = []
for size in input_sizes:
    G = nx.erdos_renyi_graph(n=size, p=0.05)
    start = time.time()
    coloring = luby_graph_coloring(G)
    end = time.time()
    runtime = end - start
    runtimes.append(runtime)
    vertex_counts.append(len(G.nodes()))
    results.append((size, runtime, dict(list(coloring.items())[:10])))  # show
only first 10 vertex colorings for readability
```

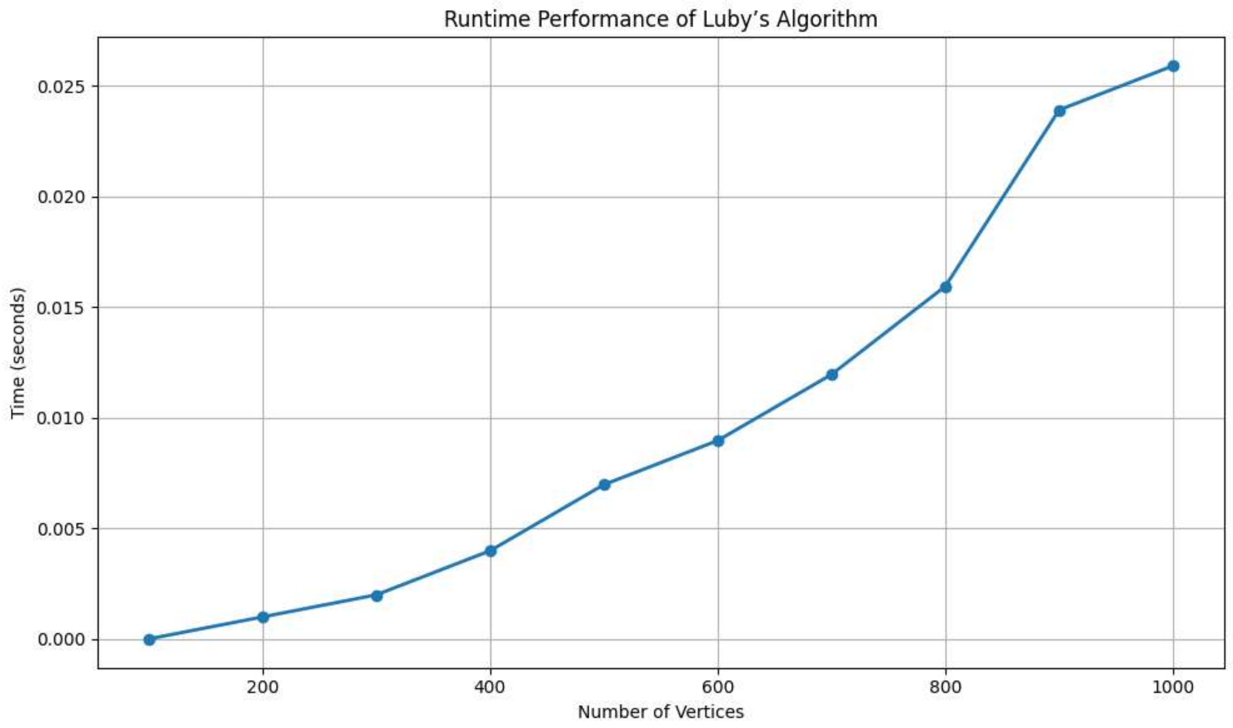
Result Analysis:

The runtime data is shown in the table below :


Luby’s Algorithm: Vertices vs Runtime Table

Vertices	Runtime (s)
100.0	0.0
200.0	0.0009965896606445312
300.0	0.001993417739868164
400.0	0.0039865970611572266
500.0	0.0069768428802490234
600.0	0.008970022201538086
700.0	0.01195979118347168
800.0	0.015946388244628906
900.0	0.02391982078552246
1000.0	0.02591419219970703

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.