Program Structure and Development

Mingzheng Michael Huang

- **Setup the Graph: (readGraph function):** reads the number of vertices and edges from a file, setting up the necessary data for the Bellman-Ford algorithm. This foundational step is critical for accurate path computation, ensuring the algorithm has the correct graph information to work with.
- **Initialization: (Initialization function):** prepares the distance and predecessor arrays with initial values, setting the stage for the Bellman-Ford algorithm. This step is vital for starting the algorithm with valid initial states, ensuring that each vertex is correctly considered during the pathfinding process.
- **Parallel Bellman-Ford Algorithm: (bellmanford function):** the core Bellman-Ford algorithm is implemented with OpenMP for parallel edge relaxation. This parallelization significantly enhances the algorithm's performance, and the inclusion of an early termination condition adds efficiency by avoiding unnecessary iterations when no further updates are possible.
- **Outputs (outputResult function):** takes care of formatting and printing the shortest paths, offering flexibility to provide results for either a specific destination or the entire graph. This function is user-centric, delivering the final outcomes of the algorithm's pathfinding in a clear and readable format.
- **Memory Handling (cleanup function):** handles memory management by freeing allocated resources. This step is crucial for preventing memory leaks and ensuring proper resource management, maintaining the program's integrity and efficiency.

Automated Validation with Bash Script - Correctness

Validation Approach

- *Sequential Baseline: Implemented Bellman-Ford in Python as a reliable baseline for correctness.*
- *Parallel Implementation: Executed parallel Bellman-Ford in C.*
- *Automated Comparison: Used a Bash script to compare the output of both implementations.*

Key Points

- *Methodology: Ran both implementations on identical graph inputs, then compared shortest path distances and paths.*
- *Rationale: Ensures parallel implementation correctness by matching output against a trusted sequential version.*
- *Results: Consistent outputs between the Python and C programs indicate correct parallelization in C.*

```
[redacted]:~/desktop/6010/hw5test$ ./compare_outputs.sh
Usage: ./compare_outputs.sh <filename> <num_threads> [destination_vertex]
[redacted]:~/desktop/6010/hw5test$ ./compare_outputs.sh graph50-1K.txt 1 1
Comparing outputs...
C Program Output:
1: 1.28920; 0 6 49 2 1

Python Program Output:
1: 1.28920; 0 6 49 2 1

Outputs differ!
```

```
[redacted]:~/desktop/6010/hw5test$ python3 bellmanford.py graph10-70.txt 1
0: 0.00000; 0
1: 6.70040; 0 8 2 1
2: 2.22850; 0 8 2
3: 4.71420; 0 3
4: 5.01520; 0 8 9 4
5: 4.85580; 0 3 5
6: 0.30450; 0 6
7: 4.72100; 0 8 2 7
8: 1.50500; 0 8
9: 3.29530; 0 8 9
[redacted]:~/desktop/6010/hw5test$ ./bellmanford graph10-70.txt 1
0: 0.00000; 0
1: 6.70040; 0 8 2 1
2: 2.22850; 0 8 2
3: 4.71420; 0 3
4: 5.01520; 0 8 9 4
5: 4.85580; 0 3 5
6: 0.30450; 0 6
7: 4.72100; 0 8 2 7
8: 1.50500; 0 8
9: 3.29530; 0 8 9
0.00001
```

Automated Validation with Bash Script - Performance

Performance Evaluation Method

- *Objective: Evaluate scalability and efficiency with varying threads (1, 2, 4).*
- *Approach: Automated Bash script to measure execution times on consistent graph input.*

Key Insights

- *Scalability: Observed performance improvements with increased threads.*
- *Efficiency: Demonstrated effective resource utilization in parallelization.*

Confirmed scalability and efficiency of the parallel implementation, as evidenced by decreased execution times with more threads.