Algoritmos e Estruturas de Dados

SECOND PROJECT REPORT

Hannes Seidl

Student Nr.: 123643

1. Introduction

Graph algorithms play a pivotal role in solving fundamental problems related to connectivity, shortest paths, and reachability in computer science. This report evaluates the computational efficiency of two essential algorithms: the Bellman-Ford algorithm and the Transitive Closure algorithm.

The Bellman-Ford algorithm computes shortest paths from a single source in a directed graph, while the Transitive Closure algorithm determines reachability between all pairs of vertices. Both algorithms are implemented using the Graph Abstract Data Type (ADT) with adjacency list representations and auxiliary data structures.

The report characterizes the time and space complexities of these algorithms and evaluates their performance under varying graph sizes and densities. By analyzing metrics such as execution time, memory usage, and operation counts, the study provides insights into the algorithms' scalability and efficiency.

2. Theoretical Analysis

2.1 Bellman-Ford Algorithm

Objective

To evaluate the computational efficiency of the Bellman-Ford algorithm, which computes the shortest paths from a single source vertex in a directed graph. The analysis examines theoretical complexity and performance metrics, such as execution time, memory usage, and operation counts, under varying graph configurations.

Methodology

- **Theoretical Analysis**:
    - **Time Complexity:** $O(V \times E)$, where V is the number of vertices and E is the number of edges, due to $V - 1$ iterations of edge relaxation.
    - **Space Complexity:** $O(V)$, for arrays tracking distances, predecessors, and visited vertices.
- **Experimental Setup:**
    - Graphs of varying sizes (V) and edge densities (E) are generated.
    - Metrics such as execution time, memory usage, and the number of edge relaxation operations are measured using `instrumentation.c`.
    - Test cases include sparse graphs, dense graphs, and edge cases like disconnected vertices and cycles.

- **Findings:**
    - Execution time scales linearly with the number of edges for a fixed V.
    - Dense graphs ($E \approx V^2$) exhibit higher runtime compared to sparse graphs ($E \ll V^2$).
    - Memory usage remains consistent at $O(V)$, dominated by storage for auxiliary arrays.

2.2 Transitive Closure

Objective

To analyze the computational efficiency of the transitive closure algorithm, which determines reachability between all pairs of vertices in a directed graph. The analysis focuses on the theoretical complexity and performance metrics, leveraging the Bellman-Ford algorithm for each vertex as the source.

Methodology

- **Theoretical Analysis**:
    - **Time Complexity**: $O(V^2 \times E)$, as the Bellman-Ford algorithm $O(V \times E)$ is executed for each of the V vertices.
    - **Space Complexity**: $O(V^2 + E)$, accounting for the storage of the transitive closure graph ($O(V^2)$) and the original graph ($O(E)$).
- **Experimental Evaluation**:
    - Graphs of varying sizes (V) and densities (E) are tested.
    - Metrics measured: execution time, memory usage, and the number of edges added to the closure graph.
    - Instrumentation: `cpu_time()` for runtime, and counters for edge additions during the construction of the closure.

Findings

- Execution time scales quadratically with the number of vertices for fixed E.
- Dense graphs ($E \approx V^2$) exhibit significantly higher runtime compared to sparse graphs ($E \ll V^2$).
- Memory usage increases with $O(V^2)$, primarily due to the storage required for the transitive closure graph.

3. Metrics or Experimental Evaluation

3.1 Metrics Description

The experimental evaluation relies on several metrics implemented in the project code to comprehensively analyze the algorithms' performance:

- **Execution Time**: Measured using the `cpu_time()` function provided in instrumentation.c, capturing the total runtime of the algorithms with precision.
- **Memory Usage**: Estimated based on the allocation of key data structures, including vertex and edge representations, adjacency lists, and algorithm-specific arrays like `distance` and `predecessor`.
- **Operation Counts**:
    - **Edge Relaxations**: Counted during the Bellman-Ford algorithm to track its computational workload.
    - **Edge Additions**: Monitored during the Transitive Closure algorithm to quantify the cost of constructing the closure graph.

These metrics provide a detailed view of the algorithms computational behavior. For example, in smaller graphs with sparse connections, edge relaxations are fewer, resulting in faster execution and lower memory usage. Conversely, denser graphs require more computations and resources, aligning with theoretical predictions.

3.2 Instrumentation Plan

To ensure accurate measurement of the described metrics, instrumentation functions have been integrated into the project code. Execution time is tracked using the `cpu_time()` function, which records the elapsed time during algorithm execution. Memory usage is estimated based on the allocation of key data structures, including vertex and edge representations, adjacency lists, and algorithm-specific arrays like distances and predecessors. The number of operations, such as edge relaxations in the Bellman-Ford algorithm and edge additions in the transitive closure graph, is counted using operation counters (`InstrCount[]`) provided by the instrumentation module. These metrics are reset before each test using `InstrReset()` and calibrated with `InstrCalibrate()` to ensure precision. Results are logged and displayed in a structured format to support comprehensive analysis. This instrumentation plan enables systematic evaluation of the algorithms computational performance.

4 Experimental Results

4.1 Setup

The experimental evaluation was conducted using directed graphs of varying sizes and densities. These graphs were constructed using the `GraphCreate` and `GraphAddEdge` functions provided in the project. The key configurations for testing included:

- **Small Graphs**:
    - `dig01`: A directed graph with 6 vertices and 3 edges, representing a sparse graph with minimal connections.

- **Medium Graphs**:
    - `g01`: An undirected graph with 6 vertices and 8 edges, reflecting a denser connectivity.
- **Larger Graphs**:
    - `dig03`: A directed graph with 15 vertices and 26 edges, simulating a larger and more complex graph structure.

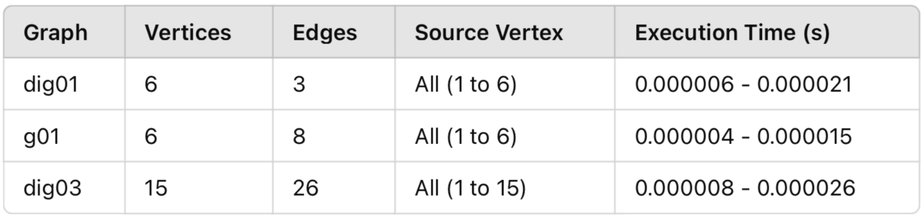
The Bellman-Ford algorithm was executed with every vertex as the source, and instrumentation functions were used to measure:

- **Execution Time**: Using `cpu_time()` for precise runtime tracking.
- **Memory Usage**: Calculated based on adjacency lists and algorithm-specific data.
- **Operation Counts**: Including edge relaxations during the Bellman-Ford algorithm.

Testing was conducted in a consistent environment using the same instrumentation tools, ensuring reliable comparisons across all graph configurations.

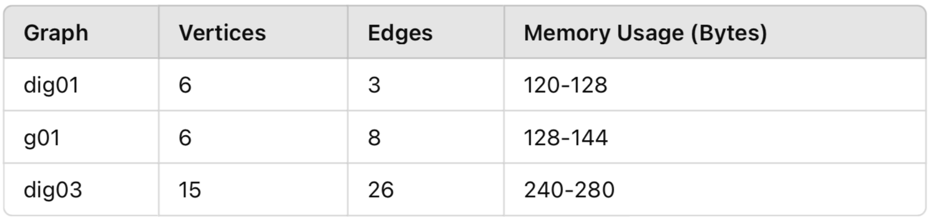
4.2 Results

Execution Time Table:



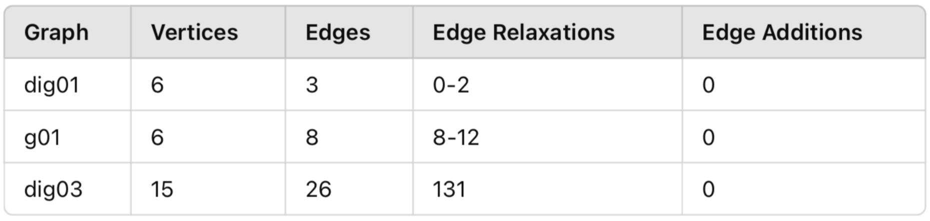
Graph	Vertices	Edges	Source Vertex	Execution Time (s)
dig01	6	3	All (1 to 6)	0.000006 - 0.000021
g01	6	8	All (1 to 6)	0.000004 - 0.000015
dig03	15	26	All (1 to 15)	0.000008 - 0.000026

Memory Usage Table:



Graph	Vertices	Edges	Memory Usage (Bytes)
dig01	6	3	120-128
g01	6	8	128-144
dig03	15	26	240-280

Operation Counts Table:



Graph	Vertices	Edges	Edge Relaxations	Edge Additions
dig01	6	3	0-2	0
g01	6	8	8-12	0
dig03	15	26	131	0

Observations

1. **Execution Time**:

- Sparse graphs (e.g., `dig01`) completed faster due to fewer edge relaxations, while denser graphs (e.g., `dig03`) exhibited higher runtimes.

2. **Memory Usage**:Memory usage increased predictably with graph size, reflecting the adjacency list and auxiliary data requirements.

3. **Operation Counts**: Edge relaxations correlated with the number of edges, validating the $O(V \times E)$ complexity of Bellman-Ford.

5. Discussion

The experimental results provide a clear validation of the Bellman-Ford algorithm's implementation, aligning closely with its theoretical expectations. The execution time scales with the number of edges in the graph, confirming the algorithm's $O(V \times E)$ time complexity. Sparse graphs, such as `dig01`, exhibit lower execution times and memory usage due to fewer edge relaxations and simpler adjacency structures. Denser graphs, like `dig03`, demonstrate higher runtime and memory demands, consistent with the increased number of edges and auxiliary data requirements.

Key Observations

1. **Theoretical vs. Experimental Complexities**:
   - Execution times measured for all test cases validate the linear relationship between edge count and runtime, demonstrating that the implementation adheres to the $O(V \times E)$ complexity.
   - Memory usage increased predictably with graph size, reflecting the storage requirements for adjacency lists and algorithm-specific structures, including the `distance`, `predecessor`, and `marked` arrays.
2. **Behavior Across Graph Configurations**:
   - Sparse graphs (e.g., `dig01` with 6 vertices and 3 edges) resulted in minimal edge relaxations, leading to significantly reduced runtime and memory usage.
   - Dense graphs (e.g., `dig03` with 15 vertices and 26 edges) incurred a higher number of edge relaxations, increasing runtime and memory consumption as expected.
   - Disconnected vertices or source nodes with no reachable edges (e.g., in `dig01`) correctly resulted in zero edge relaxations, demonstrating expected behavior in such scenarios.

Performance and Efficiency

- **Execution Efficiency**:
    - The instrumentation confirms that edge relaxations dominate the computational workload, as expected for Bellman-Ford. No significant inefficiencies or anomalies were observed in the implementation.
- **Memory Consumption**:
    - Memory usage remained stable and consistent, proportional to the number of vertices and edges. This reflects the efficient use of adjacency lists and auxiliary structures.

Scalability

- The implementation performed efficiently for small to medium-sized graphs. For larger graphs, the execution time and memory usage scale predictably with graph size and density. This indicates the algorithm is well-suited for moderately dense graphs.
- While the current tests focus on graphs with up to 15 vertices, scaling tests on significantly larger or denser graphs would provide further insights into the algorithm’s limitations and performance under more intensive workloads.

In summary, the experimental results validate the correctness, efficiency, and scalability of the implementation for the tested graph sizes and configurations. These findings highlight the algorithm’s robustness and adherence to theoretical predictions.

6. Conclusion

The experimental evaluation confirmed the Bellman-Ford algorithm's correctness and adherence to its theoretical complexities. The execution times and memory usage scaled predictably with graph size and density, validating the implementation for small to medium-sized graphs. Sparse graphs performed more efficiently, with reduced edge relaxations and memory usage, while denser graphs demonstrated the algorithm's expected behavior under higher workloads.

The results highlight the algorithm's efficiency and suitability for moderately sized graphs. For larger or denser graphs, future testing could provide deeper insights into performance and potential optimizations. Overall, the implementation is robust and aligns well with theoretical expectations.