Proposal

Jiewen Hu
https://github.com/Gudora/15418-project

December 4, 2023

1 Summary

I am going to implement a coarse-grained, fine-grained, and lock-free graph data structure based on linked-list and implement basic graph related algorithm like DFS and BFS based on it.

2 Background

Graph data structures are fundamental in modeling relationships between entities in various domains, such as networks, VLSI design, and graphics. Represented as an ordered pair $G = (V, E)$, these structures are essential in applications that require dynamic modifications, including insertion and deletion of vertices and edges. With the advent of multi-core systems, the focus has shifted towards designing efficient concurrent data structures. Concurrent dynamic unbounded graphs, in particular, are well-suited for multi-threaded implementations on multi-core computers, enhancing a broad spectrum of applications.

A key operation in graph algorithms is the reachability query, which involves determining if a path exists between two vertices. In dynamic and concurrent graph environments, where vertices and edges can be concurrently modified, ensuring the correctness and validity of these queries poses significant challenges. This is further complicated by operations like vertex deletion, which involves removing the vertex and its connected edges, potentially impacting concurrent operations.

3 Challenges

The most significant challenges in parallelizing graph algorithms lie in handling the intricate dependencies and dynamic nature of graph structures. Graph operations, such as path finding or updating connections, depend heavily on the current state of the graph. This state is prone to frequent changes, especially in a concurrent setting where multiple threads might modify vertices and edges simultaneously, leading to potential data inconsistencies and race conditions.

Additionally, graph algorithms often exhibit irregular memory access patterns due to their non-linear data structures. This irregularity poses challenges in optimizing cache usage and memory efficiency, as accessing graph elements like vertices and edges does not follow a predictable pattern. The divergent execution paths, especially evident in depth-first or breadth-first searches, add to the complexity, as different threads may need to traverse vastly different parts of the graph, making load balancing and efficient parallelization a challenging task.

Addressing these specific issues in graph algorithms will be a focal point of the project, aiming to enhance both the understanding and implementation of efficient, concurrent graph processing.

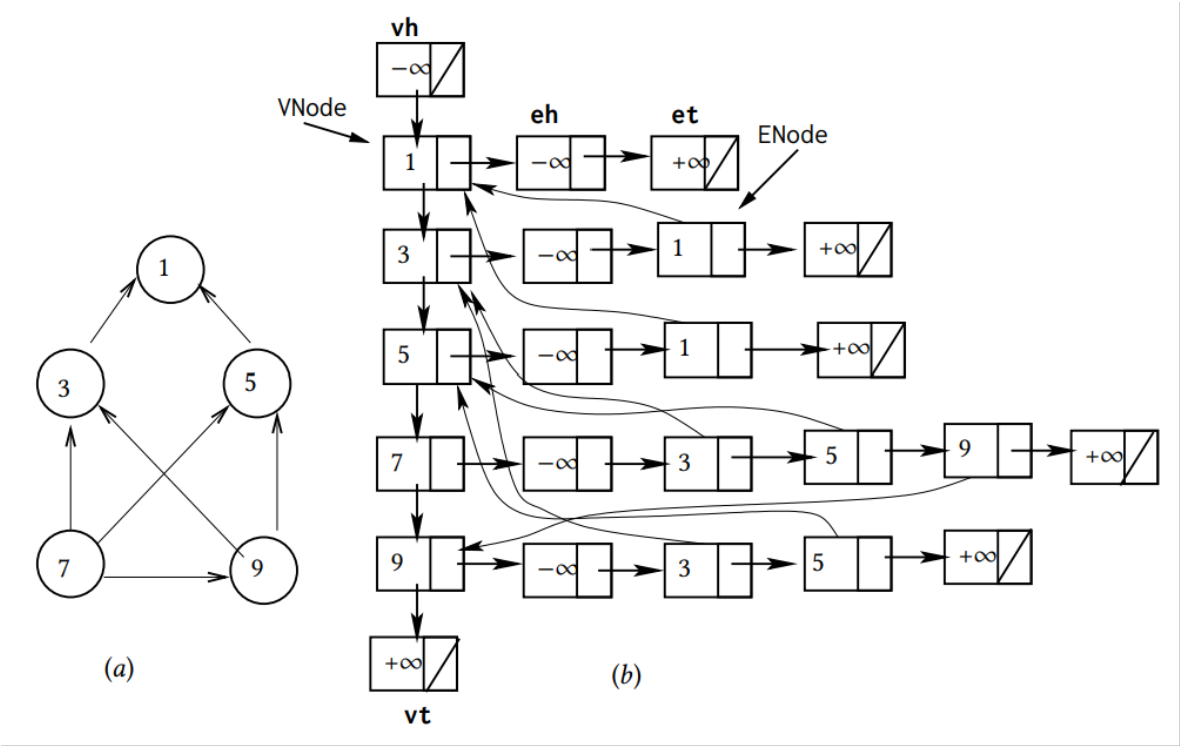

Figure 1: Composite linked list representation of a graph.

4 Resources

The primary resources are related papers. Here are some I found might be quite useful:

1. Chatterjee, B., Peri, S., Sa, M., & Singhal, N. (2018). A Simple and Practical Concurrent Non-blocking Unbounded Graph with Linearizable Reachability Queries. arXiv preprint arXiv:1809.00896.

2. Peri, S., Reddy, C. K., & Sa, M. (2019). An Efficient Practical Concurrent Wait-Free Unbounded Graph. IEEE Xplore. https://ieeexplore.ieee.org/document/8855669

3. Hong, B. (2008). A Lock-free Multi-threaded Algorithm for the Maximum Flow Problem. Drexel University, Philadelphia, PA 19104. ISBN 978-1-4244-1694-3. ©2008 IEEE.

5 Goals

5.1 Plan to achieve

1. Coarse-Grained Graph: Implement a basic graph structure with a coarse-grained locking mechanism. This version will serve as a foundational model for understanding and comparing more advanced implementations.

2. Fine-Grained Graph: Develop an advanced version of the graph using fine-grained locking. This implementation aims to improve concurrency control by reducing the granularity of locks, thereby allowing more parallel operations.

3. Lock-Free Graph: Create a sophisticated lock-free graph implementation. This version will utilize non-blocking algorithms to manage graph operations, focusing on maximizing parallel efficiency and reducing contention.

4. Efficient Reachability Queries: Implement and optimize reachability queries within the graph structures, especially in the lock-free version, ensuring they are efficient and accurate even in a highly dynamic and concurrent environment.

5. Performance and Scalability Analysis: Conduct thorough performance testing and scalability analysis across all graph implementations. This involves assessing how each version performs under different workload types, graph sizes, and levels of concurrency.

5.2 Hope to achieve

1. Dynamic Graph Optimization: Enhance the lock-free graph to efficiently handle dynamic changes such as concurrent additions and deletions of nodes and edges without compromising performance or data integrity.

2. Parallel Graph Algorithms Integration: Integrate and evaluate parallel versions of complex graph algorithms, such as minimum spanning tree, within the lock-free graph structure.

6 Platform

I will implement different versions of the graph in C++ and test them on the GHC machines.

7 Timeline

Week Starting	**Primary Goals**
November 13	Begin implementation of Coarse-Grained Graph
November 20	Continue with Coarse-Grained Graph; Start Fine-Grained Graph implementation
November 27	Complete Fine-Grained Graph; Begin Lock-Free Graph implementation
December 4	Finalize Lock-Free Graph; Optimize Efficient Reachability Queries
December 11	Explore advanced goals

Table 1: Weekly Progress Plan for Concurrent Non-blocking Graph Algorithm Project