Rao Zhang (zhangrao@stanford.edu)

Title

Implementation and Exploration of Rust-based Graph Processing Library

Summary

- Design and implement a graph processing library in rust
- Compare different approaches to memory management
- Evaluate the effects on usability and performance

Background

- Graph is a very common concept and widely used in various applications in various areas.
- Graph processing library aims to deal with problems related to graphs conveniently and efficiently.
- Rust is such a powerful language with fast speed as well as multiple choices of memory management.

Challenges

- Learn and implement the whole library with the new grammars and collections in Rust.
- Understand and correctly apply concepts of various smart pointers in Rust.
- Design the graph processing library and make it efficient in time and space.

Resources

- Starter code: assignment 6 RIIR
- Tutorial: Rust official documentation
- Comparison/reference: CS106B graph library

Goals and deliverables

- An unweighted graph library based on Rust with basic functionalities
  Basic methods: addEdge, addNode, getEdges, getNeighbors, getValue, getEdgeSet, getNodeSet, isConnected, isEmpty, removeEdge, removeNode, size, toString, diameter
- A weighted graph library based on Rust with advanced functionalities (if time permits)
  Algorithm implementation such as shortest path, minimum spanning tree, cycle detection.
- A final report
  Document the issues during development.
  Show the memory performance comparison between different approaches.
  Show the performance difference with other graph libraries.

Schedule

- Nov 06 --- Nov 12
  Learn about Rust memory management and graph processing library.
- Nov 13 --- Nov 19
  Design and implement the graph processing library in Rust.
- Nov 27 --- Dec 03
  Design and implement the graph processing library in Rust.
  Try different methods of memory management in Rust.
- Dec 04 --- Dec 10
  Make performance comparison with C++ graph processing library in CS106B.
  Write the final report.