Byung Hun Lee

CS 408

Raheja

CS408 Project 1 Report

My application is based on the Bron-Kerbosch algorithm. The Bron-Kerbosch algorithm find and list all subsets of vertices where each pair of vertices in a subset is connected by an edge, and no subset can have any additional vertices added to it while preserving its complete connectivity. In simpler terms, find the maximal cliques in an undirect graph, where clique is a complete subgraph of G and maximal clique is a clique that is not a subset of a larger clique. According to Wolfram Alpha the Bron-Kerbosch algorithm is an efficient method for finding all maximal cliques in a graph despite other similar algorithms existing. The algorithm I implemented finds the maximal clique when given three sets and finds the maximal clique that includes all the vertices in R, some of the vertices in P, and none in X. The basic goal of finding maximal clique is to add the union of P and X (or sets b and c in my code) to every element of R (set a in my code). When P and X are both empty, R is basically a maximal clique.

Each of my four implementations (ADT and non-ADT in both Java and C++) ran the algorithm a hundred times with the same graph. The graph I used had 3 vertices called 0, 1, and 2, with an edge between 0 and 1, 1 and 2, and 2 and 0; basically, a simple triangle. My first test was on a non-ADT implementation on Java. The total time for a hundred loop was 0.015846 second, with an average run time of 0.000158 second. The second test was using ADT on Java, resulting in a total run time of 0.00454 second and an average run time of 4.543e-5 second; basically 3 times as slow as the non-ADT implementation. In the non-ADT implementation in C++, the total time was 0.3419 seconds and an average time of 0.00342 seconds. Compared to the Java implementation, the C++ version was approximately 20 times as slow. Finally, the ADT implementation in C++ had a total time of 0.04697 second and an average time of 0.0004697. Compared to the difference between the non-ADT implementation between Java and C++, the difference in time between Java and C++’s ADT implementations was comparatively less, with only a 10-time difference in speed.

Based on just the speed it would be easy to assume that Java was faster than C++, but it does not take into consideration the difference in implementation. I did try my best to mirror the implementation between Java and C++ to make time comparison more realistic; however, I believe part of the reason for the huge difference might be due to the difference in code between Java and C++. Still, considering the difference in speed between the two languages, I believe part of the difference in speed lies in with how the two-programming language operates. Still, considering how there was a 20-time difference between Java and C++ in non-ADT, yet only a 10-time difference between Java and C++ in ADT, one of two or both must be true. My implementation for ADT was better in C++ compared to non-ADT or C++ performed better due to how the linkedlist was created, using structure as nodes in the same class as sets as opposed to how linkedlist and nodes were created in Java.