CS 3353 – Spring 2020 – The Online Edition

Programming Assignment 3 – Kruskal's MST Algorithm

Due: April 13 2020 at 11:59 pm.

Overview

In this project, you'll implement Kruskal's algorithm for finding the minimum spanning tree of a graph.

Disjoint Set Data Structure[1]

Implementation of Kruskal's algorithm requires management of a *collection of disjoint sets*. A disjoint-set data structure contains a set of unique elements (like a set), but where these elements are partitioned into disjoint (non-overlapping) subsets.

For example, consider the following set of elements:
{1, 2, 5, 6, 9, 10, 12, 14, 18}

The following would represent a disjoint-set:
{[1, 2, 9], [5, 10, 12], [6], [14, 18]}

However, this organization of data would not because 6 and 10 appear in two different subsets:
{[1, 2, 5],[6], [9, 6, 10], [10, 12, 14, 18]}

The major operations on a disjoint-set data structure are:

- **makeSet**(someVal) – creates a new set in the collection with someVal
- **find**(x) – returns some identifier of the set of the collection that contains the element x.
- **union**(a, b) – creates a new set in the collection that is the union of the sets containing elements a & b.

Example Operations and Effects on the Container

Operation:	Contents of disjoint set collection:
-	{}
makeSet(10)	{[10]}
makeSet(15)	{[10], [15]}
makeSet(20)	{[10], [15], [20]}
makeSet(25)	{[10], [15], [20], [25]}
find(15)[2]	{[10], [15], [20], [25]}
union(15, 25)	{[10], [15, 25], [20]}
union(15, 20)	{[10], [15, 25, 20]}

1 Also called the Union-Find data structure

2 Returns some unique identifier such that if compared to the result of separate find(15) call would result in a finding of equivalence.

The trivial implementation of a disjoint set data structures is a set of linked lists. As part of this project, you'll implement this trivial version as well as a more sophisticated version based upon independent research.

Your Tasks

1. Implement a templated disjoint set data structure. You should have two versions that share the same interface which is inherited from a super class:
   a) Trivial implementation using a set of linked lists
   b) More sophisticated implementation based upon independent research you will do
2. Implement Kruskal's algorithm on a graph read in from a file. The output should consist of the set of edges that are part of the MST and the sum of the weights of the edges of the MST. Your implementation should follow the general pseudocode as it appears in the MST PDF from lecture.
3. Generate (or locate) sample graphs of increasing size (number of nodes) and density (number of edges compared to the number of nodes). Analyze how Kruskal's algorithm performs when the underlying disjoint set uses linked lists or your more sophisticated implementation.

What You Will Submit

- Full source code implementation of your project following best practices in design and implementation.
  - The result of building and executing the project you submit must **completely reproduce**[3] from top to bottom the data used in analysis for point 3 below.
  - The run process should not require any command line arguments to execute.
  - All output should be to one or multiple files that are easily recognized by the user (Morgan and me).
- A set of at least 10 test graphs that are used in your performance analysis[4]
- A short analysis paper not to exceed 2 typed, single-spaced, 10 pt font (times or arial) pages with maximum 1" margins. No more than 25% of the paper can be consumed by graphs.

Implementation Notes

- We are not dictating the input file format. You may devise your own or use a standard such as GML (Graph Modeling Language) for example.
- We are not dictating the output file format. You may write the output to files in your own format with the only restriction of being able to match one input graph with the output of processing that graph.
  - This doesn't mean it can be completely disorganized and require some amount of mind-reading. I should be able to look at the set of input files and easily determine which output file (or section in the output file) goes with a particular input file.
- You should make use of compiler optimization flags where useful.

3 For more info on reproducible research, see https://www.displayr.com/what-is-reproducible-research/ or https://lib.colostate.edu/services/data-management/reproducible-analysis/.

4 If you are familiar with Python, you might want to explore the NetworkX library for sample graph generation.

Point Allocation for Grading

Total Points: 100

Disjoint Set – Trivial Implementation	5 points
Disjoint Set – Robust Implementation	10 points
Proper Kruskal’s Implementation	20 points
Code Quality	10 points
Clarity of Building and Execution	15 points
Analysis Paper	40 points