Minimum Spanning Tree Algorithms

CS 375 Final Project

Tim Hung and Samuel David Bravo

Binghamton University

May 3, 2016

Overview

Minimum Spanning Trees

A minimum spanning tree connects all the vertices in a graph together into a tree with the lightest weight possible.

Approach

Algorithms:

- Kruskal's
- Prim's

Implementations:

- Adjacency List
- Adjacency Matrix

Problem Statement

How do Prim's and Kruskal's algorithms handle graphs of different densities?

Do they depend on whether the graph is implemented as an adjacency list or an adjacency matrix?

Prim's Algorithm

Main Idea

Expand the tree by adding the lightest connecting edge.

Pseudocode

```
EdgeContainer MST = empty
VerticesContainer keys = graph.vertices
for (v in keys) v.distance = infinity
keys[0].distance = 0

while (keys.someone_not_in_tree()) {
    Vertice v = keys.get_smallest()
    keys.add_to_tree(v)
    keys.update_distances(graph[v].neighbors)
    MST += v.edge
}
```

Implementation Details

Analysis of Prim's

interesting features, time complexity, why?

Kruskal's Algorithm

Main Idea

- Separate vertices into disjoint sets
- Reorder all edges by smallest weight first
- Loop through edges, add it to tree if its vertices are disjoint

Pseudocode

```
EdgeContainer all_edges = graph.sorted_edges()

VerticesSet set = disjoint_set(v.size)
EdgeContainer MST = empty

for (Edge e : all_edges)
    v1 = e.source;
    v2 = e.destination
    if (set.are_vertices_disjoint(v1,v2))
        MST += e;
        set.join(v1,v2)
```

Key Data Structures

- A vector of all the sorted edges
- A vector for holding edges included to minimum spanning tree
- A vector representing disjoint sets

Functions

- ReadFromAdjacencyList or ReadFromAdjacencyMatrix
- SortEdgesSmallestFirst, size $|E| \log ||E|$
- CreateDisjointSets, size $|V|$
- JoinSets, size $|V|$
- AreSetsDisjoint, size $O(1)$

Analysis of Kruskal's

- Only uses graph representation for retrieving edges
- Sorting the edges is of size $|E| \log |E|$
- Meanwhile, looping through edges is of size $|E|$
- The time complexity is carried by the sort, $|E| \log |E|$
- Kruskal's works well with sparse graphs

Results

Demonstration

Our Data

Describe the dataset that you used to test the algorithm. How did you generate it? What characteristics does it have, and why? What did you decide to vary in the input set, and why?

Results

What did you learn from testing your algorithm?

Limitations and Future Work

What limitations does your project currently exhibit? If you had another month, what could you improve? What additional tests would you run?

- Kruskal's can finish early by checking if there is only 1 set. If that's the case, the for loop will finish in $|V|$ instead of $|E|$.

Summary

Recap

This is a recap of what we have talked about.

Questions

Thank you.
Any questions?