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
}
```

Key Data Structures

- A vector of the vertices with the following information: {weight to tree, nearest tree vertice}
- A vector for holding edges included to minimum spanning tree

Functions

- InitializeVertices, with a run time of $|V|$
- ExtractMin, with a run time of $lg|V|$
- UpdateDistances, with a run time of $|E|$

Analysis of Prim's

- Prim's is dependent on analyzing all of the nodes
- Extracting the minimum vertices has a running time of $|V|\lg|V|$
- Updating the distances has a running time of $|E|lg|V|$
- Prim's running time is of the form $O(|V|lg|V|) + O(|E|\lg|V|)$
- Prim's is suitable for dealing with dense graphs.

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

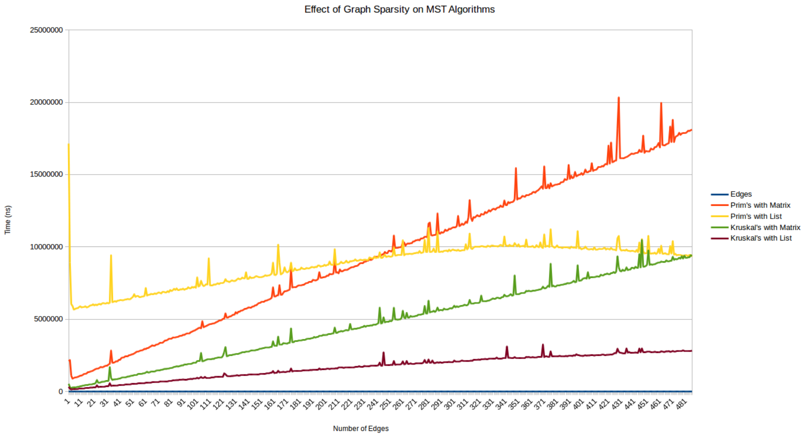
- ReadFromAdjacencyList or ReadFromAdjacencyMatrix
- SortEdgesSmallestFirst has a running time of $|E| \lg ||E|$
- CreateDisjointSets has a running time of $|V|$
- JoinSets has a running time of $|V|$
- AreSetsDisjoint has a running time of $O(1)$

Analysis of Kruskal's

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

Results

Data and Results



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 $|E|$.

Questions

Thank you.
Any questions?