Minimum Weight Spanning Tree Algorithms

Benjamin Case

3/4/19

There are three main algorithms used for computing minimum weight spanning algorithms. We give a brief description of each, and we implement and analyze Sollin's algorithm.

Prims:

- Data structures used: min heap
- The idea is to start with an arbitrary vertex v and add its (outgoing) edges to a min heap. Then remove the min edge (v,u) and add mark the node u as part of the MST, add all of u's outgoing edges that go to nodes not yet marked as part of the MST to the min heap.
- Prims: https://www.youtube.com/watch?v=Uj47dxYPow8
- $O(nd \log_d n + m \log_d m)$ when using d-heaps

Kruskal's:

- Data structures used: disjoint set, min heap
- The idea is to let each vertex be a disjoint set and place all the edges in a min heap. Remove the minimum edge from the heap, if it connects to disjoint sets, union those sets and add the edge to the MST, if the edge does not connect to disjoint sets discard the edge. Continue until only one set left.
- Kruskal's: https://www.youtube.com/watch?v=Yo7sddEVONg
- Running time: O(m log n)

Sollin's/Boruvka's:

- Data structures used: disjoint set, min heaps
- Kind of a cross between Prims and Kruskals.
- First let each vertex be a disjoint set. The to each disjoint set create a min heap of edges leaving that tree. Loop through the disjoint sets (trees) and choose the min edge from its min heap, take the new reached vertex and union its set with the present one and merge their corresponding min heaps. Continue looping through the disjoint sets until there is only one set.
- Sollin's algorithm: https://www.youtube.com/watch?v=t92xyTDvl_c
- Running time: O(m log n)

Performance Comparison: Kruskl's generally performs worse than either of the other two. As graphs grow large, Prims performs better on dense graphs than Sollins, and Sollins performs better than Prims on sparse graphs.

Time and Space Analysis for our Sollin's implementation

The asymptotic performance of Sollin's is O(log n) times through the outer loop. Overall then this gives O(m log n) performance. For our implementation of Sollin's we used a leftist heap, a disjoint set data structure and a hash table. We have not done a full amortized analysis of these data structures or their effect on our implementation of Sollin's. We have followed standard implementations of these, so we would expect their performance to match what is necessary to maintain the O(m log n) performance.

As for space the algorithm we hold all the edges in heaps and a list of all the vertices, so the space is linear in the graph size, i.e. O(m + n). We have not had time to do a precise calculation to determine the constants.

Empirical Timing Analysis.

Here is the table of the timings we have gathered, where **n** is the number of vertices and **m** is the number of edges in the graph.

average times in sec					
m \ n	500	1000	2000	5000	10000
2000	0.0077	0.011851			
4000	0.0079	0.013365			
6000	0.0075	0.014266			
8000	0.0072	0.014035			
10000	0.0076	0.013497			
12000	0.0075	0.01406	0.02883638	0.0493	
14000			0.02912999		
20000			0.02898037		
30000			0.02905112		
40000			0.0308805		
50000			0.030389186		0.131126166

References

For our implementation we consulted and followed the following resources.

- Leftist heap:
  - pseudo code in Tarjan pg 39 (pdf pg 49).
  - C++ implementation https://www.geeksforgeeks.org/leftist-tree-leftist-heap/
- Disjoint set:
  - pseudo code in Tarjan pg 24 (pdf pg 34).
  - Goodrich, Tamassia, Goldwasser's textbook *Data Structures and Algorithms in Python* has a Python implementation that we used https://doc.lagout.org/programmation/python/Data%20Structures%20and%20Algorithms%20in%20Python%20[Goodrich,%20Tamassia%20&%20Goldwasser%202013-03-18].pdf

For timing asymptotics we referred to the survey paper:

Bazlamaçcı, Cüneyt F., and Khalil S. Hindi. "Minimum-weight spanning tree algorithms a survey and empirical study." *Computers & Operations Research* 28.8 (2001): 767-785.