Long Project 2 Report : Prim 1, Prim 2, Kruskals and Edmonds Algorithm

Detailed Analysis : Edmonds Algorithm

Description:

Edmonds Algorithm for Minimum Spanning Tree identification on Directed Graph believes in finding the minimum incoming edge into a node in order to make incremental progess towards ideal MST it can find. Whenever it comes across unreachable edges, it finds a zero weight cycle, shrinks and repeats the process until it finds a Zero Weight MST.

Detailed Analysis : Optimization Process:

Since this algorithm involves a lot of computations and originally is of Order O(E*V), it was crucial to choose data structures which help reduce the time complexity as well as space complexity of the default algorithm.

Since the algorithm heavily depends on retreival of edges from the node based on the edge weight, priorityQueue data structure was used in order maintain edgelist as reverseEdgeList.

Also, PseudoVertex inner class was created in order to efficiently manage creation and entire Life cycle of Cycles which are shrunk into a Pseudo-Vertex. Optimal Incoming Edge as well as the cycle excluding the duplicate edge to the “a” node (to which the optimal edge from outside c enters” were computed and associated with the pseudoVertex during creation of the PseudoVertex itself.

Usage of Priority Queue wherever Minimum-edge retrieval was required, boosted performance of the Algorithm Drastically. Also, the Default implementation of Graph was modified in order to not store edges in Adjacancy and RevAdjacecny List as they were not required anymore.

Since the algorithm is Recursive and loops over same tree again and again, once we have found the repeating vertex, we reuse the stack in order to avoid having to traverse entire tree just to find out the exact path travelled.

Please find Statistics for each of the following files below. As we can see for the huge input file of size(100000 105000), (~17mb input), the readGraph executes in 4 seconds and MST computation happens in 1 minute 42 seconds.

Statistics

lp2-data\0-lp2.txt

Statistics for Step => input graph read statistics :
Time: 25 msec.
Memory: 3 MB / 192 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 2 msec.
Memory: 3 MB / 192 MB.

Total Weight of the MST was found to be : 17
(5,2)
(4,3)
(1,4)
(3,5)

lp2-data\1-lp2.txt

Statistics for Step => input graph read statistics :
Time: 30 msec.
Memory: 4 MB / 192 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 5 msec.
Memory: 4 MB / 192 MB.

Total Weight of the MST was found to be : 180922

lp2-data\2-lp2.txt

Statistics for Step => input graph read statistics :
Time: 37 msec.
Memory: 4 MB / 192 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 10 msec.
Memory: 4 MB / 192 MB.

Total Weight of the MST was found to be : 178316

lp2-data\3-lp2.txt

Statistics for Step => input graph read statistics :
Time: 53 msec.
Memory: 5 MB / 192 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 19 msec.
Memory: 6 MB / 192 MB.

Total Weight of the MST was found to be : 244483

lp2-data\lp2-ck.txt
Statistics for Step => input graph read statistics :
Time: 317 msec.
Memory: 95 MB / 243 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 11843 msec.
Memory: 351 MB / 935 MB.

Total Weight of the MST was found to be : 4964126

lp2-data\lp2-m.txt
Statistics for Step => input graph read statistics :
Time: 4332 msec.
Memory: 489 MB / 827 MB.

Statistics for Step => Edmonds : Minimum Spanning Tree retrieval :
Time: 102468 msec.
Memory: 965 MB / 1286 MB.

Total Weight of the MST was found to be : 50324411

Prim 1 & Prim 2 Algorithm Analysis

As we can see, the Space complexity of the Prim 1 algorithm is O(E) where as space complexity of Prim 2 is O(V).

Since O (E) can actually be equal to $O(V^2)$, Prim 2 algorithm is much better way of implementing the Minimum spanning tree algorithm Although for small trees, output of both the algorithms look identical, we can see that as size of the tree increases, the time taken by Algorithm 2 definitely stands out as compared to Algorithm 1. We can see that the Prim 2 algorithm is more than four times faster than prim 1 algorithm. For Larger inputs, Prim 2 performs way better than Prim 1 would.

Please find statistics for Prim 1 and Prim 2 below.

Statistics for 10000 nodes and 9070678 edges

Prim 1 Statistics:
Statistics for Step Minimum Spanning Tree Function :
Time: 1935 msec.
Memory: 628 MB / 1353 MB.

Weight of the tree found to be 9999

Prim 2 Statistics
Statistics for Step Minimum Spanning Tree Function :
Time: 577 msec.
Memory: 1181 MB / 1407 MB.

Weight of the tree found to be 9999

Conclusion

As we can see, Prims Algorithm 2 implementation achieves much better time complexity as we are associating the index of the vertex in heap with vertex itself. This operation helps us retrieve index with O(1) time complexity. Thus Prims algorithm 2 is much better choice of implementation than prims algorithm 1.

Kruskals Algorithm:

Key idea behind Kruskals algorithm is to start creating components with single representative and merging components which do not have same representatives but have and edge in common.

Optimization Steps:

After Implementing the initial implementation, we took following steps in order to tune it further.

We noticed that we were reading the input graph and storing edges in adjecancyList as well as revAdjecancyList - this was not really required for Kruskal's Algorithm. Kruskal's algorithm required edges to be fetched by weight - least ones first. By going ahead with base implementation of graph, we were using much more space than what was required. Each edge had four copies - one in adjescecnyList and one in revAdjescencyList duplicated in both vertices u and v.

In order to make the space complexity one fourth, if only we could optimize the initial load of the graph itself, we would not only save on space complexity (O(4E)-> O(1E)) but also save a lot on initial sorting of the Edges which is the first step of kruskal's. Use of Priority Queue on Edge defined in Graph class helped achieve this. Also, a small change to signature of Graph.readGraph lead of making it support more functionality with minimal changes.

Also, Find method was implemented using a simpler alternative to recursion to avoid performance overhead.

Statistics for Step Minimum Spanning Tree Function using kruskals algorithm is:
Time: 1 msec
Memory: 2MB/128MB.

Weight of the tree found to be 12.