Report: Analysis of Topological Sort

A graph is formally defined as the ordered pair G = (V, E) where V is a set of nodes and E is a set of edges. It would thus be a good strategy to look at what effects changing V and E have on our algorithms for Topologically Sorting a graph using Depth First Search and Kahn's Algorithm.

***Note:** Here on, we will use |V| to signify the number of vertices and |E| to signify the number of edges. All the values plotted on the graphs are averages taken over five inputs for the same combination of | V | and | E|. We use clock ticks in order to measure the running time of our algorithms on various inputs.*

|V| is constant, |E| is variable: I kept |V| constant at 2000 and used daggen.c to vary the probability of getting an edge between two nodes from 0 to 100. As we can see from the graph that topological sorting using DFS and Kahn's method are both linear with respect to the number of edges, O(|E|) complexity.

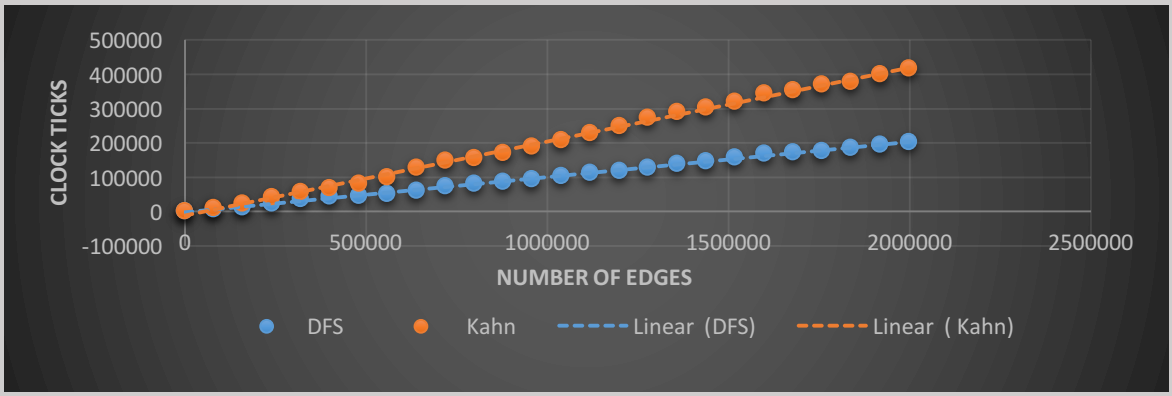


Figure 1: |V| constant, |E| variable

|E| is constant, |V| is varying: In order to vary |V|, I fixed |E| at a value of 10000 and varied |V| from 1000 to 27000 in steps of 1000. The results indicate that Topological Sorting, irrespective of the algorithm is linear with respect to the number of vertices i.e. O(|V|).

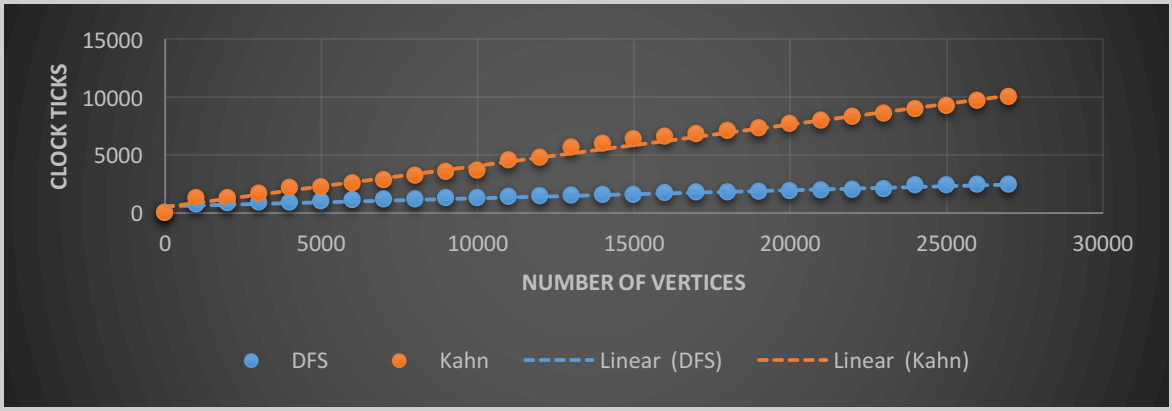


Figure 2: |E| constant, |V| variable

|V| and |E| both varying: We now try to show that Topological Sort is linear even when we randomize |V| and |E| in order to change them simultaneously. As we can see from the plot clearly the Topological sort is linear with respect to |V| + |E|, so it's complexity is O(|V| + |E|). As we can see from all the graphs that using DFS is much faster than using Kahn's method even though both are linear

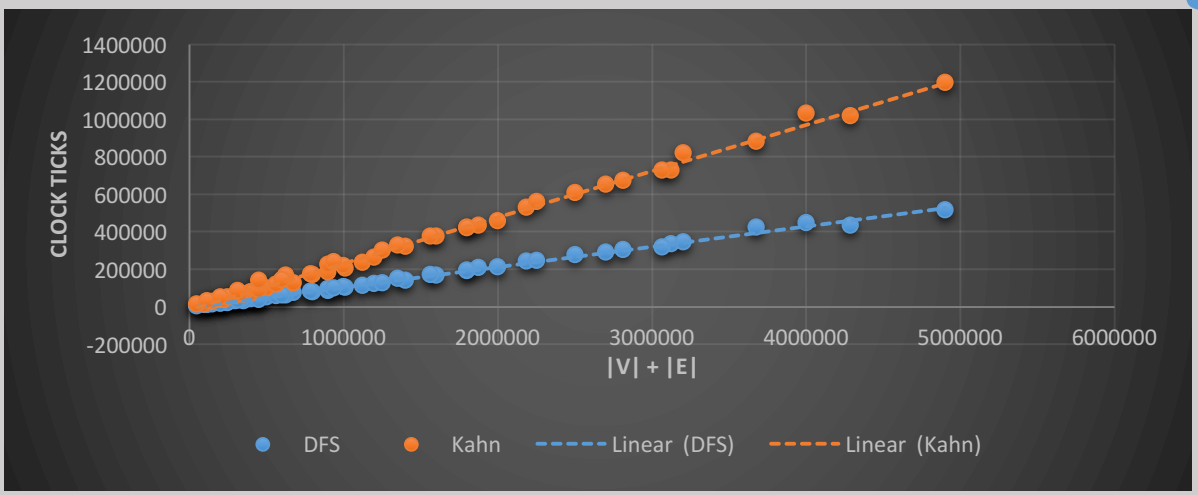


Figure 3: |E| and |V| variable

Making sense of the findings:

Looking into the DFS algorithm, we mark each vertex as being unvisited in O(|V|) time and then start visiting each vertex using the visit function (we only visit a vertex if it has not been visited before). We propagate down the graph (using the outgoing edges of the vertices) until there are no more vertices to visit at which point we return back to a vertex which allows us to find an alternate path. We continue this process till we find a back edge (causing exit since we cannot topologically sort a cyclic graph) or if we have finished visiting all the vertices. Since we visit each vertex only once and each edge of each vertex is traversed once (as a consequence of each vertex being visited only once) we can conclude that the algorithm has complexity of O(|V| + |E|).

The analysis for Kahn Sort is a little bit more complicated. We start by constructing a list of vertices with no incoming edges in O(V) time. We then pop the head of the list of source vertices, add it to the tail of our sorted vertices and start deleting it's outgoing edges. If any of the vertices that are being pointed to turn into source vertices due to the deletion of the edge we add it to the head of the list of source vertices. We repeat the above process until the time the list of source vertices is empty. Now the sorted list so obtained is the topologically sorted list if the input was a DAG. We check if a graph was valid by checking if we deleted all of it's edges. Here we assume that insertion to the tail of a list and deletion of an edge is O(1) time which is not true so in order to preserve the complexity of the algorithm we simulate the deletion of an edge by the decrement of a counter which stores the number of incoming edges to the vertex, and we add new elements to the sorted list at the head instead of the tail. Both of our alternate operations are O(1) time and the cost of O(|V|) in order to reverse the sorted list is also bearable. Now since we put every vertex into the list of source vertices and pop it, we will run the loop |V| times, moreover we 'delete' each edge of each vertex once so our algorithm is linear in |V| + |E| with complexity O(|V| + |E|).

Kahn VS DFS:

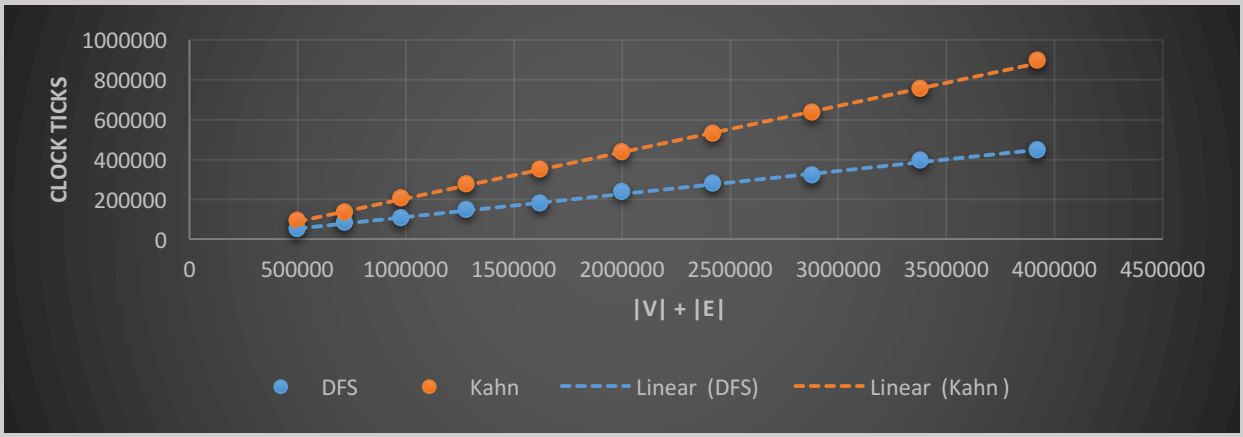
A comparison of two algorithms is warranted if they are designed to execute the same task. DFS sort has a few advantages over Kahn sort, for example in case we have a cyclic graph as input DFS would find it as soon as we encounter a back edge but Kahn sort would execute the whole program create a topologically sorted list only to find the invalidity of the input thereafter. Moreover, DFS tends to be much faster than Kahn as input size increases. For example, DFS in twice as fast as Kahn for the values given in Table 1 of Appendix A.

Thus, we can conclude through analysis and experimentation that Topological Sort using DFS and Kahn's algorithm are linear with complexity O(|V| + |E|).

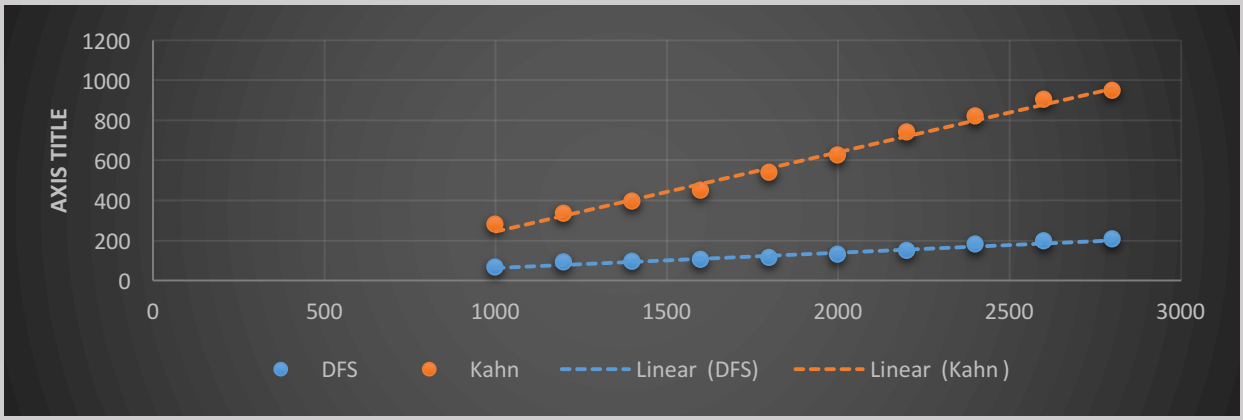
APPENDIX A:

Pathological Inputs:

Graph 1: 100% edge probability given to daggen.c for different vertex sizes:



Graph 2: 0% edge probability given to daggen.c for different vertex sizes:



Both the extreme cases above show linear behavior enforcing my claim that both DFS and Kahn's sort are linear with respect to |V| and |E| if implemented with care.

Table 1: (Time measured in clock ticks)

|V| + |E|	100675	200651	300261	400116	500500
DFS	7453	17640	30971	39594	50889
Kahn	12517	35791	55956	76703	99000

APPENDIX B:

References used in project:

- Wikipedia(https://en.wikipedia.org/wiki/Topological_sorting)
- Introduction to Algorithms, CLRS