CSc 3102: Topological Sorting

A DFS-based Strategy

- Reversed Depth-First Search-based Topological Sorting
- Reversed Out-degree-based Topological Sorting
- In-degree-based Topological Sorting

1 Introduction

Tasks with dependencies can be naturally modeled using a directed graph without any directed cycle. The linear ordering of the tasks in such a way that if task u must be done before task v then u comes before v in the linear ordering. There are two strategies in the textbook but we will give the pseudocode description of only the depth-first search-based strategy in this course. We will briefly discuss the other strategies during the lecture.

2 Topological Sort

The strategy is to perform an out-directed depth-first traversal keeping track of the order in which the vertices are visited. The topological sort labeling is the reverse order of the depth-first search traversal. There are also in-degree-out-degree strategies for performing topological sort that we will discuss.

We now present a pseudocode description of the reverse dfs-based topological sorting algorithm.

```
ALGORITHM: TopoSort(D,linorder)
   {Input: D - a directed graph with n vertices
          and vertex set V <- {1,2,...,n}
   linorder: a linear ordering of the vertices of the directed
             graph such that no parent vertex succeeds its
             child vertex in the ordering.}
   Seen[1..n] <- false
   count <- n
   for v <- 1 to n
      if Seen[v] = false
         DFSOut(D,v,Seen,linorder)
      endif
   endfor
endAlgorithm

Algorithm: DFSOut(D,v,Seen,linorder)
   Seen[v] <- true
   for each w in V such that vw is in E
      if Seen[w] = false
         DFSOut(D,w,Seen,linorder)
      endif
   endfor
   linorder[count] <- v
   count <- count - 1
endAlgorithm
```

Analysis:
TopoSort on directed graph is $O(n+m)$ and $O(n+2m)$ on simple graph. So it is $O(n+m)$.

We can also topologically sort the vertices of a digraph by successively removing vertices with zero out-degree (no out-directed arc) until all the vertices are removed from the digraph. Each time a vertex is removed, all its incident edges are deleted from the graph. The vertices are listed in reversed order as they are removed.

Alternatively, We can topologically sort the vertices of a digraph by successively removing vertices with zero in-degree (no in-directed arc) until all the vertices are removed from the digraph. Each time a vertex is removed, all its incident edges are deleted from the graph. The vertices are listed in the order that they are removed from the digraph. The algorithms based on the in-degree-out-degree strategy are both $O(n^2)$. Why? These algorithms for topological ordering are examples of decrease-and-conquer algorithms. Why?