Topological sorting: pseudocode and analysis

CSCI 382, Algorithms

September 20, 2019

Definition 1 *A* topological ordering *(*topological order, topological sort, topsort*) of a directed graph is an ordering of nodes $v_1, \ldots, v_n$ such that for every $(v_i, v_j) \in E$, we have $i < j$.*

In class we proved that a directed graph G has a topological ordering if and only if G is acyclic. Our proof was "by algorithm", but the algorithm was somewhat naïve: at each step, do a $\Theta(V)$ search to find a vertex with indegree 0, add it to the ordering, and delete it from the graph. This algorithm has a worst-case running time of $\Theta(V^2)$ overall. Notice this has nothing to do with the number of edges.

However, we can do better, especially if there are fewer than $\Theta(V^2)$ edges. The idea is to keep track of the current set of vertices with indegree 0 so we don't have to search for one at each iteration. We can create this set initially with some precomputation, and then keep it updated efficiently during the algorithm. In order to keep it updated efficiently, we also have to keep track of the current indegree of each vertex.

Pseudocode is shown on the next page. How long does this take?

- Lines 2–4 each take $\Theta(1)$.
- Initializing *in* (lines 5–7) takes $\Theta(V + E)$ since we loop over all vertices and then look at every edge once.
- Initializing Z (lines 8–10) takes $\Theta(V)$, assuming we can add to Z in $\Theta(1)$ time.
- The main loop will execute $\Theta(V)$ times; in each loop we do some $\Theta(1)$ operations (remove from Z, append to T), and do $\Theta(1)$ work for each of several edges. Again, we will look at each edge once in total, so this main loop takes $\Theta(V + E)$ time.

Overall, then, this algorithm is $\Theta(1) + \Theta(V + E) + \Theta(V) + \Theta(V + E) = \Theta(V + E)$.

Require: G is a directed acyclic graph (DAG)

```
1:  function TopSort(G)
2:      T ← empty list                                  ▷ T stores the topsort
3:      Z ← empty queue/stack/whatever                  ▷ Z stores vertices with indegree 0
4:      in ← dictionary mapping all vertices to 0       ▷ in stores current indegree of each vertex
5:      for each v ∈ V do                               ▷ initialize in
6:          for each u adjacent to v do
7:              increment in[v]
8:      for each v ∈ V do                               ▷ initialize Z
9:          if in[v] = 0 then
10:             add v to Z
11:     while S is not empty do                         ▷ main loop
12:         v ← Z.remove
13:         append v to T                               ▷ get next vertex for the topsort
14:         for each u adjacent to v do                 ▷ update in and Z
15:             decrement in[u]
16:             if in[u] = 0 then
17:                 add u to Z
        return T
```

Figure 1: TopSort(G)