Lecture 13: Searching III: Topological Sort

Lecture Overview: Search 3 of 3

- BFS vs. DFS
- job scheduling
- topological sort
- strongly connected components

Readings

CLRS, Sections 22.4 and 22.5 (at a high level)

Recall:

- Breadth-First Search (BFS): level by level
- Depth-First Search (DFS): backtrack as necessary.
- Both $O(V + E)$ worst-case time $\implies$ optimal
- BFS computes shortest paths (min. # edges)
- DFS is a bit simpler & has useful properties
**Job Scheduling:**

Given Directed Acyclic Graph (DAG), where vertices represent tasks & edges represent dependencies, order tasks without violating dependencies

![Dependence Graph](image)

**Source**

Source = vertex with no incoming edges

= schedulable at beginning \((A,G,I)\)

**Attempt**

BFS from each source:

- from A finds \(H,B,C,F\)
- from D finds \(C,E,F\) need to merge - costly
- from G finds \(H\)

![BFS-based Scheduling](image)
Topological Sort

Reverse of DFS **finishing times** (time at which vertex’s outgoing edges finished)

We have a new field `time` that stores the finishing time. To get a topological sort that solves the job scheduling problem, we simply run the DFS procedure below.

```python
def dfs_visit(V, Adj, s):
    for v in Adj[s]:
        if v not in parent:
            parent[v] = s
            dfs_visit(V, Adj, v)
    ft = ft + 1
    time[s] = ft

def topsort(V, Adj):
    parent = {}
    for s in V:
        if s not in parent:
            parent[s] = None
            dfs_visit(V, Adj, s)
```

Given the `time` dictionary, one can generate all keys from the dictionary and insert into an array of length \(|V|\) indexed by the appropriate finishing time `ft`.

In Figure 1, we run `dfs_visit` starting from vertex A and reach B, C and F. F finishes first, followed by C and B in the recursion. Next, we reach H from A. Then we are done with A. `dfs_visit` beginning with A generates a depth-first tree with the vertices A, B, C, F, and H, and the edges \((A, B), (B, C), (C, F), \) and \((A, H)\). We next run `dfs_visit` starting with vertex D and reach vertex E. (Vertices C and F have already been visited.) This generates the depth-first tree with vertices D and E and with edge \((D, E)\). We next start and end with vertex G, since we have already explored H. This generates the depth-first tree with the vertex G and no edges. Finally we start and end with vertex I. This generates the depth-first tree with the vertex I and no edges.

The reverse order of the finishing times shown in Figure 1 is a topological sort.

Note that the DFS procedure can be run on any graph – the graph does not have to be a DAG. We can compute finishing times for each vertex. These will depend on the order edges are listed in `Adj`. Even if the graph is not a DAG, these finishing times are useful. In particular, they are useful in determining the strongly connected components (SCCs) of a graph.
Strongly Connected Components

$C$ is an SCC of a directed graph $G(V, E)$ if for every pair of vertices $u$ and $v$ in $C$ there is a path from $u$ to $v$ and a path from $v$ to $u$. The SCCs of a DAG correspond to the vertices, i.e., each vertex is an SCC. For graphs with cycles, SCCs are non-trivial to compute.

For an algorithm to compute the SCCs of a graph, see CLRS Second/Third Edition 22.5. You should understand the algorithm, but you are not responsible for the proof of correctness.