Topological Sort

CSE 373
Data Structures
Lecture 19

Readings and References
• Reading
  ✔ Section 9.2

Some slides based on CSE 326 by S. Wolfman, 2006

Problem: Find an order in which all these courses can be taken.
Example: 142 → 143 → 378 → 370 → 321 → 341 → 322 → 401

In order to take a course, you must take all of its prerequisites first

Given a digraph $G = (V, E)$, find a linear ordering of its vertices such that:
for any edge $(v, w)$ in $E$, $v$ precedes $w$ in the ordering

Any linear ordering in which all the arrows go to the right is a valid solution

Note that F can go anywhere in this list because it is not connected.
Not all can be Sorted

- A directed graph with a cycle cannot be topologically sorted.

Cycles

- Given a digraph $G = (V, E)$, a cycle is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that
  - $k < 1$
  - $v_1 = v_k$
  - $(v_i, v_{i+1})$ in $E$ for $1 \leq i < k$.
- $G$ is acyclic if it has no cycles.

Topo sort algorithm - 1

**Step 1:** Identify vertices that have no incoming edges
- The "in-degree" of these vertices is zero

Topo sort algorithm - 1a

**Step 1:** Identify vertices that have no incoming edges
- If no such vertices, graph has only cycle(s) (cyclic graph)
- Topological sort not possible – Halt.

Topo sort algorithm - 1b

**Step 1:** Identify vertices that have no incoming edges
- Select one such vertex

Topo sort algorithm - 2

**Step 2:** Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.
Continue until done

Repeat Step 1 and Step 2 until graph is empty

Select

Repeat Step 1 and Step 2 until graph is empty

Select B. Copy to sorted list. Delete B and its edges.

Select C. Copy to sorted list. Delete C and its edges.

Select D. Copy to sorted list. Delete D and its edges.

Select E. Copy to sorted list. Delete E and its edges. Select F. Copy to sorted list. Delete F and its edges.

Remove from algorithm and serve.

Done
### Implementation

Assume adjacency list representation

Translation array: `1 2 3 4 5 6`

### Calculate In-degrees

```java
for i = 1 to n do
    D[i] := 0
end for
for i = 1 to n do
    x := A[i];
    while x ≠ null do
        D[x.value] := D[x.value] + 1;
        x := x.next;
    endwhile
end for
```

### Maintaining Degree 0 Vertices

**Key idea:** Initialize and maintain a queue (or stack) of vertices with In-Degree 0

### Topo Sort using a Queue

After each vertex is output, when updating In-Degree array, enqueue any vertex whose In-Degree becomes zero

### Topological Sort Algorithm

1. Store each vertex’s In-Degree in an array D
2. Initialize queue with all “in-degree=0” vertices
3. While there are vertices remaining in the queue:
   (a) Dequeue and output a vertex
   (b) Reduce In-Degree of all vertices adjacent to it by 1
   (c) Enqueue any of these vertices whose In-Degree became zero
4. If all vertices are output then success, otherwise there is a cycle.
Some Detail

Main Loop
while notEmpty(Q) do
  x := Dequeue(Q)
  Output(x)
  y := A[x];
  while y ≠ null do
    D[y.value] := D[y.value] - 1;
    if D[y.value] = 0 then Enqueue(Q,y.value);
    y := y.next;
  endwhile
endwhile

Topological Sort Analysis

- Initialize In-Degree array: O(|V| + |E|)
- Initialize Queue with In-Degree 0 vertices: O(|V|)
- Dequeue and output vertex:
  » |V| vertices, each takes only O(1) to dequeue and output: O(|V|)
- Reduce In-Degree of all vertices adjacent to a vertex and Enqueue any In-Degree 0 vertices:
  » O(|E|)
- For input graph G=(V,E) run time = O(|V| + |E|)
  » Linear time!