Directed Graph Algorithms – Topological Sort

CSE 373
Data Structures
Winter 2007

Readings

• Reading Chapter 9
  › Section 9.2

Problem: Find an order in which all these courses can be taken.
Example: 142 → 143 → 378 → 370 → 321 → 341 → 322 → 326 → 421 → 401

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

Topological Sort

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.
Also the solution is not unique.

Any linear ordering in which an arrow goes to the left is not a valid solution.

NO!
Paths and Cycles

• Given a digraph G = (V,E), a path is a sequence of vertices v₁, v₂, …, vₖ such that:
  › (vᵢ, vᵢ₊₁) in E for 1 ≤ i < k
  › path length = number of edges in the path
  › path cost = sum of costs of each edge

• A path is a cycle if:
  › k > 1; v₁ = vₖ
• G is acyclic if it has no cycles.

Only acyclic graphs can be topologically sorted

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

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 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.

Done
**Implementation**

Assume adjacency list representation

<table>
<thead>
<tr>
<th>Translation array</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Calculate In-degrees**

In-Degree array; or add a field to array A

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

**Maintaining Degree 0 Vertices**

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

```
Queue
```

**Topo Sort using a Queue (breadth-first)**

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

```
Queue
```

**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|)
  \[ \text{Linear time!} \]

Topo Sort using a Stack (depth-first)

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