MCS-375: Algorithms: Analysis and Design Handout #GF4
San Skulrattanakulchai
Gustavus Adolphus College Nov 10, 2015

Directed Acyclic Graphs (DAGS)

A dag is a digraph containing no cycle. Every dag can be drawn in such a way that all edges are directed down (or from left to right).

[EXAMPLE PICTURE HERE]

Dags model many real-world problems, e.g., combinational circuits, prerequisite digraphs, dependency digraphs, Bayes nets/belief systems, ...
A vertex in a dag is called a *source* (*sink*) if no edge enters (leaves) it, i.e., it has indegree (outdegree) 0.

Lemma. *Every dag has at least one source (sink).* □

Topological sort
A *topological numbering* of a digraph is an assignment of an integer to each vertex so that each edge is directed from lower number to higher number.

EXAMPLE HERE

Theorem. *A digraph has a topological numbering if and only if it is acyclic.*

Proof. $\Rightarrow$: Any path in a topologically numbered digraph goes from lower to higher number. Thus, every path is simple. Hence, no cycle can exist.
$\Leftarrow$: Assign the highest number n to some sink s. Then delete s and recursively number the remaining graph. □

The proof suggests the following high-level algorithm:

repeat until G has no vertices
  grow a dfs path P until a sink s is found
  set $T[v]$ to n, reduce n by 1, and delete s from P and G

Implementation

Use the array $T[1..n]$ for two purpose.

$$T[v] = \begin{cases} 0 & \text{if } v \text{ has never been on any dfs path} \\ t & \text{if } v \text{ has been deleted and assigned toplogical number t} \end{cases}$$

```
TOP(G) {
  num ← n
  for each vertex v do
    T[v] ← 0
  for each vertex v do
    if T[v] = 0 then
      dfs(v)
}
DFS(v) {
  for each edge (v, w) do
    if T[w] = 0 then
      dfs(w)
  /* v is now a sink in the high level algorithm */
  T[v] ← num
  num ← num − 1
}
```

Timing

TOP(G) takes time $O(m + n)$.

Notes

(i) Our algorithm shows that numbering the vertices in decreasing finish time gives a topological numbering.

Question Would numbering the vertices in order of increasing discovery time gives a valid topological numbering?

(ii) Another topological sort algorithm works by repeatedly finding a source s, assigning the next lowest number to it, and then deleting s from the current graph.

This algorithm can also be implemented in linear time.

Algorithms on dags

Suppose each edge uv has a nonnegative length $\ell[u, v]$. We can find a longest path in G in $O(m + n)$ time.

Idea: We'll set $d[v]$ to the length of a longest path starting at v. We'll compute $d[v]$ values for all v **in reverse topological order**, using the formula

$$d[v] = \max\{\, 0, \ell[v, w] + d[w] : (v, w) \in E \,\}$$

We calculate the values specified by the formula by modifying the dfs code as follows.

```
LONGEST(G) {
  for each vertex v do
    d[v] ← −1
  for each vertex v do
    if d[v] = −1 then
      dfs(v)
}
DFS(v) {
  d[v] ← 0
  for each edge (v, w) do {
    if d[w] = −1 then
      dfs(w)
    d[v] ← max{ d[v], ℓ[v, w] + d[w] }
  }
}
```

Notes

(i) The longest paths in a dag are known as "critical paths" when doing *critical path scheduling*.

(ii) Finding a longest path in a general graph is NP-complete.

(iii) Our algorithm illustrates dynamic programming on dags. Similar algorithms can be used to calculate the longest path from s to t or shortest paths from a vertex s, etc.