Data Structures and Algorithms

Spring 2009-2010

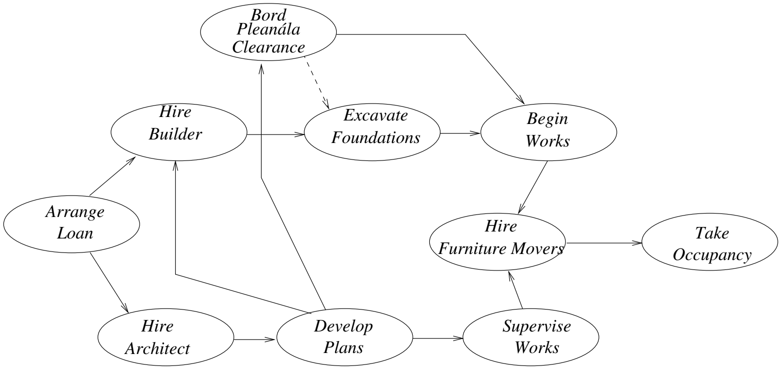
Outline

Outline

Introduction

- In a directed graph it is easy to tell what vertices precede a given vertex, v (just check all adjacency lists for v)
- We often want to generalize this to build a list for *every* vertex of what vertices precede it
- That is, we want to sort the graphs by precedence or, *topologically*
- Key property required for topological sort (TS): the graph must be *acyclic*
- There is not a *unique* ordering following TS: only guarantee is that if w is a descendant of u then w occurs later than u in ordering

Introduction (contd.)



- Call the *indegree* of a node the number of edges entering the node

TopSort Algorithm

Repeat $n = |V|$ times:

- Find a node with indegree$= 0$
- Record this node
- Remove it and any of its edges

```
void topsort( graph g )
{
  for( int ct = 1; ct <= vert_ct; ct++ )
  {
    vertex v = find_vertex_of_indegree_zero( );
    if (v == null ) fail( "graph has a cycle" );

    top_num[ v ] = ct;
    for each w adjacent to v  // edge (v, w)
      indegree[ w ]-;
  }
}
```

TopSort Algorithm (contd.)

- Problem with this code is the $O(n^2)$ running time due to $O(n)$ time required to find a zero-indegree vertex, as each vertex will surely become
- Can improve as follows:
- For every edge (v, w) we delete, we can check if w's indegree becomes 0
- If yes, store w in special DS (queue or stack)
- Instead of trawling through all $O(n)$ vertices for one of zero-indegree, just take one from queue / stack

➜ Requires an initial search to put all known zero-indegree vertices of graph on some data structure

- Running time is now $O(|V| + |E|)$ since each edge is processed exactly once and initial checking for indegree of 0 costs $O(n)$
- Note check for cycles in graphs

TopSort Algorithm (contd.)

```
void topsort( graph g )
{
  unsigned int ct = 1;
  Stack<vertex> stk( vert_ct );

  for each vertex v
    if (indegree[ v ] == 0 ) stk.push( v );

  while( !stk.is_empty( ) )
  {
    top_num[ v = stk.pop( ) ] = ct++;
    for each w adjacent to v
      if (-indegree[ w ] == 0 ) stk.push( w );
  }
  if (ct <= vert_ct ) error("graph has a cycle");
}
```